Report No. SFIM-AEC-ET-CR-95093

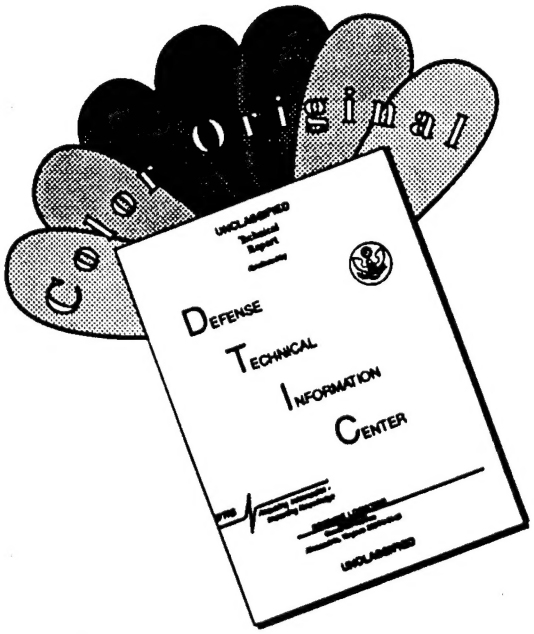
U.S. Army Environmental Center

DETECTION AND DISCRIMINATION TECHNIQUES FOR TOTAL FIELD MAGNETOMETERS AND MULTI-AXIS GRADIOMETERS (FINAL REPORT)

2 NOVEMBER 1995

Prepared by Arete Engineering Technologies Corporation

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF COLOR PAGES WHICH DO NOT REPRODUCE LEGIBLY ON BLACK AND WHITE MICROFICHE.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	2 November 1995	August 1994 - July 1995

4. TITLE AND SUBTITLE

Detection and Discrimination Techniques for Total Field Magnetometers and Multi-Axis Gradiometers (Final Report)

5. FUNDING NUMBERS

6. AUTHOR(S)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Explosive Ordnance Disposal Technology Division
Project Engineer: Arnold Burr 301/743-6850
2008 Stump Neck Road
Indian Head, Maryland 20640-5070

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Environmental Center
Project Officer: Kelly Rigano 410/612-6868
SFIM-AEC-ETP
Aberdeen Proving Ground, Maryland 21010-5401

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

SFIM-AEC-ET-CR-95093

11. SUPPLEMENTARY NOTES

Supporting Contractor: Arete Engineering Technologies Corporation
1725 Jefferson Davis Highway
Suite 707
Arlington, Virginia 22202

Development of detection and discrimination techniques for total field magnetometers and multi-axis gradiometers was performed by Arete Engineering Technologies Corporation of Arlington, Virginia. This report summarizes the work performed under contract N00174-94-C-0061.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Unlimited Distribution; Approved for Public Release

12b. DISTRIBUTION CODE

"A"

13. ABSTRACT (Maximum 200 words)

A project investigating detection and discrimination techniques for total field magnetometers and multi-axis gradiometers is being performed under the Unexploded Ordnance (UXO) Clearance Technology Program sponsored by the U.S. Army Environmental Center and managed by the Naval Explosive Ordnance Disposal Technology Division. The project was developed to address the Government's need for reliable, efficient, and cost-effective UXO clearance technology. This need has become a priority within the Department of Defense because of military downsizing, reductions, and base consolidations, and the need to close, clean-up, and return military bases and ranges to the public.

Arete Engineering Technologies Corporation developed an automatic processor for use on the Subsurface Ordnance Characterization System which has an array of cesium vapor magnetometers. The processor is a stand alone computer code, written in Ansi C, which automatically (without an operator) detects and characterizes magnetic anomalies. The prototype processor performs five major sub-tasks: (1) data input and mapping, (2) noise pre-processing, (3)threshold detection, (4) parameter estimating, and (5) target output information. It serves as a baseline algorithm for future software efforts.

This report discusses the major program algorithms, their theoretical bases, and overall program logic flow. In addition, processor upgrade paths are identified for near and long term efforts.

14. SUBJECT TERMS

unexploded ordnance, automatic processor, magnetometers, subsurface ordnance characterization system

15. NUMBER OF PAGES
133

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

VA-064-064-95-TR

DETECTION AND DISCRIMINATION TECHNIQUES FOR TOTAL FIELD MAGNETOMETERS AND MULTI-AXIS GRADIOMETERS

FINAL REPORT

2 NOVEMBER 1995

D. DeProspo
R. DiMarco

Arete Engineering
Technologies Corporation
1725 Jefferson Davis Highway, Suite 707
Arlington, Virginia 22202

This page is intentionally left blank

TABLE OF CONTENTS

This page is intentionally left blank

1.0 INTRODUCTION

Under this contract, Areté Engineering Technologies Corporation (AETC) has developed and demonstrated an automatic processor for SOCS (Subsurface Ordnance Characterization System) magnetometer data. The SOCS array is being developed by NAVEODTECHDIV. The processor is a stand alone code, written in ANSI standard C for portability with unlimited distribution rights. The code accepts standard SRD data structures as input, and automatically (without an operator) detects and characterizes magnetic anomalies. Output of target characteristics is in standard STD structure for direct use by downstream program elements. The AETC automatic processor, *Magproc*, has been validated on SOCS test data collected at Tyndall Air Force Base.

The AETC automatic processor was developed using the following methodology: first, a physics based model was identified which represented the response of a total field magnetometer to UXO targets of interest. During this time effort was also devoted to characterizing the underlying sensor noise field which generally contains significant contributions from both spatial and temporal sources. Once the sensor signal and noise fields were understood, algorithms were developed to automatically detect significant magnetic anomalies in sensor data. Algorithms were then developed to optimally extract target parameters for each anomaly using the physical signal model.

This report is organized into two main sections. In the first, major program algorithms are discussed in detail and, where appropriate, the theoretical bases for these algorithms are examined. In addition the overall program logic flow is described and examples are given, using SOCS test data, which illustrate intermediate sequential data processing steps. In the second section, processor upgrade paths are reviewed and separated into near-term and long-term realizability. In addition to the aforementioned sections, four appendices are included in the report. Appendix A provides instructions for compiling and linking the *Magproc* program, Appendix B includes a full alphabetized source code listing for reference, Appendix C contains a sample target STD file and Appendix D contains logic diagrams for the clustering and maximum likelihood parameter estimation algorithms.

This page is intentionally left blank

2.0 ALGORITHM AND LOGIC FLOW OVERVIEW

The *Magproc* processor is roughly divided into five major areas which are illustrated in Figure 2.1. Each of these major sections is discussed in order of program appearance.

2.1 INPUT AND MAPPING

At the start of a processing session the user is prompted for a magnetometer filename, which contains the total field magnetometer amplitude data for each sensor; a navigation file name, which contains the GPS position of the antenna on the back of the towing tractor; and a platform filename, which provides information regarding the motion of the articulated trailer hitch. In addition the user is prompted for the name of the output STD file, which will contain the target lists and a parameter file specifying the local earth magnetic field inclination and declination. The inclination is defined as the angle between the horizontal plane and the total magnetic field vector (positive down) while the declination is defined as the angle in the horizontal plane of the total magnetic field vector with True North (positive clockwise). The inclination and declination must be specified in the format "I3 <cr> I4 <cr>" in degrees and need be accurate only to several degrees. If the inclination and declination are not specified then default values (65,0) are used. Future versions of *Magproc* will not require this information since these parameters will be calculated in an earth magnetic field model. Failure to specify the other program inputs will result in program termination. From this point on the code is completely automatic and control is passed to the input modules.

The main input modules *readmag.c*, *readnav.c*, and *readplat.c* provide an interface between SOCS (SIDCAPS) generated SRD files and local *Magproc* data structures. These local data structures, output from the above code modules, serve as input to the code module *mapper.c*, which maps data from each sensor to ground based, relative x, y coordinates, and outputs a single data structure for use in subsequent modules.

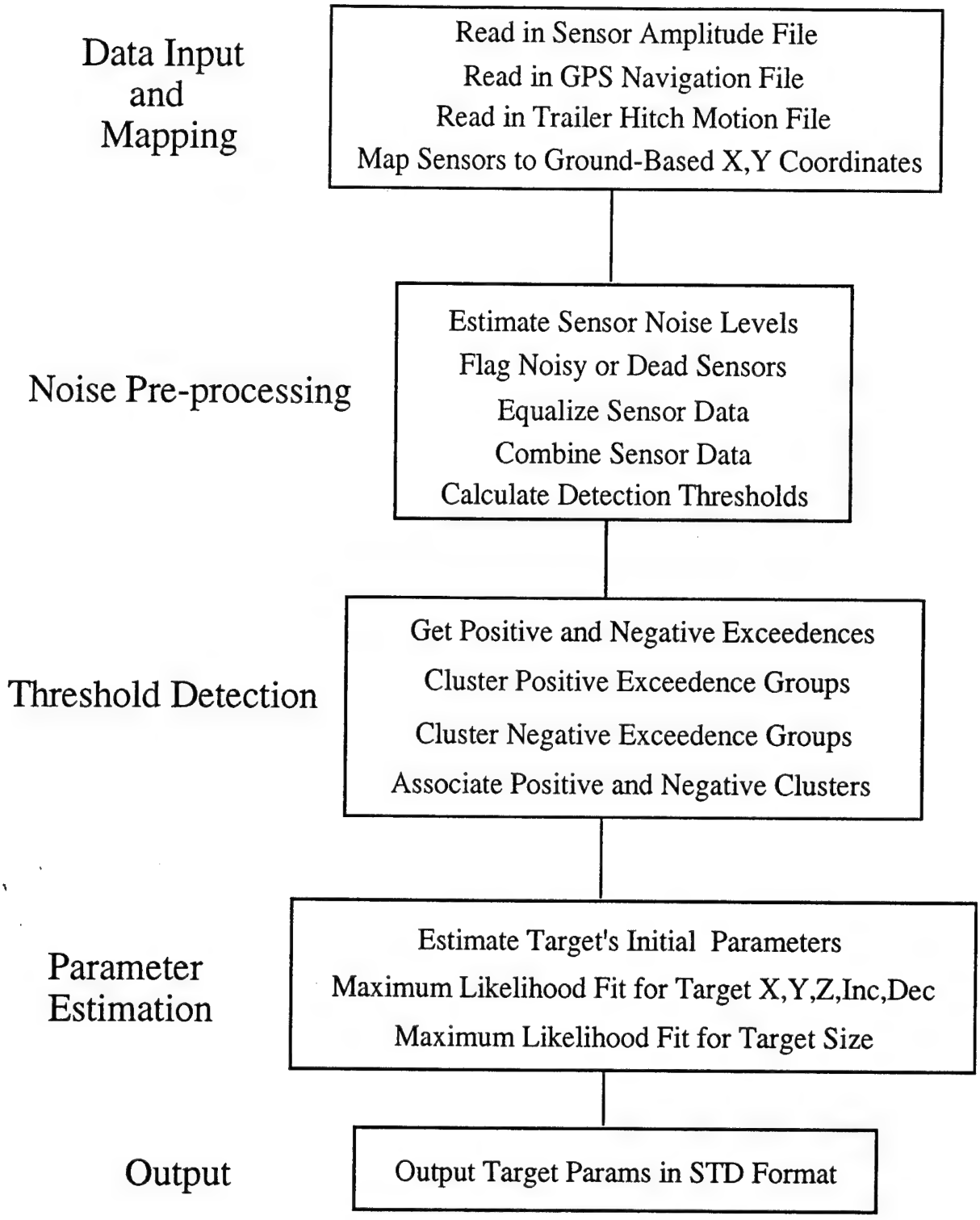


Figure 2.1 Shown are the major processing areas for the Magproc program and the primary tasks carried out in these areas.

The navigation file contains the location of the SOCS system's GPS antenna as a function of time in latitude and longitude units. For convenience inside the processor, these coordinates are converted to UTM meters, using the ***WGS 84*** datum as a standard, and a local reference point is subtracted to aid in maintaining numerical accuracy. The conversion to UTM, and later conversion back to latitude and longitude, are performed using the Cartographic Projections Procedures, written by Gerald I. Evendon. This public domain package is available from the USGS *gie@charon.er.usgs.gov*.

The platform file contains translation, roll, pitch and yaw information to determine the location of the magnetometers relative to the GPS antenna. This information is combined by time synching with the magnetometer and navigation file to map the magnetometer amplitudes to ground-based coordinates.

2.2 NOISE PRE-PROCESSING

This code area is a critical precursor to the processor target detection and parameter estimation algorithms. In it noisy or dead sensors are identified and removed from further consideration. For the remaining sensors, a dc bias is removed from the amplitude data. These algorithms depend on two statistical measures: first, an estimation of the rms noise level for each sensor; and second, an estimation of the median level for each sensor. We have found that these two quantities can be extracted from a cumulative distribution function (CDF) which provides robust estimates even in the presence of strong signal contamination if a large enough swath of data is included. **The underlying assumption is that the targets represent a non-gaussian perturbation to a gaussian sensor noise field and contribute predominately to the tails of the CDF distribution**. The median, used for dc bias removal, is extracted from the 50th percentile point of the CDF while the rms is estimated by subtracting the amplitude at the 69th percentile point from the amplitude at the 31st percentile point of the CDF. This would correspond to the plus and minus half-sigma points for a gaussian distribution. The rms levels for the magnetometers are intercompared to check for instruments with anomalously high (noisy) or low (poor sensitivity) rms. Data from identified bad sensors are ignored. Future versions of *Magproc* will produce a log file which will catalogue this data.

For each good sensor, the median is subtracted from the data. This has the effect of removing the large (50,000 nT) local earth field contribution, which can cause numerical accuracy problems, and also corrects for smaller instrument-to-instrument bias. The median subtraction is performed independently for both dominant SOCS heading directions. The directional correction is necessary because in SOCS data as much as a 10 nT shift may occur between different sensors or, in a given sensor, between data taken in the two opposite heading directions. **Note: Since the SOCS cart is driven in a staggered race-track pattern and data acquisition is turned off on turns, only the two straight trajectories are considered.** For a discussion of sensor directional bias and other SOCS system noise issues, including noise measurement and reduction, please refer to the AETC quick look report regarding SOCS performance during the JPGII demos in August and September of 1995 (Ref 1).

Once these sensor amplitude biases have been removed, and consequently, the sensors equalized, the data from the good sensors can be combined into single contiguous arrays. Then, an overall noise level can be estimated and input to the target detection routine. Figure 2.2 shows data taken at Tyndall Air Force Base in Florida with the SOCS array on an unseeded field approximately 350 by 150 ft. The data have been mapped and equalized. Figure 2.3 shows data taken at the seeded raised field (run ps2) at Tyndall Air Force Base for the NAVEODTECHDIV and WRIGHT LABORATORIES SUBSURFACE ORDNANCE CHARACTERIZATION DEMONSTRATION during April 1995. The data have also been mapped and equalized.

The primary code modules in this processing area are:

A) *Getstats.c* which calculates the rms and median statistical measures.

B) *Inscheck.c* which flags noisy or dead sensors.

C) *Combinedata.c* which performs sensor equalization, combines the data from good sensors into contiguous arrays and calculates an overall rms level for the data.

At this point program control is passed to the target detection modules.

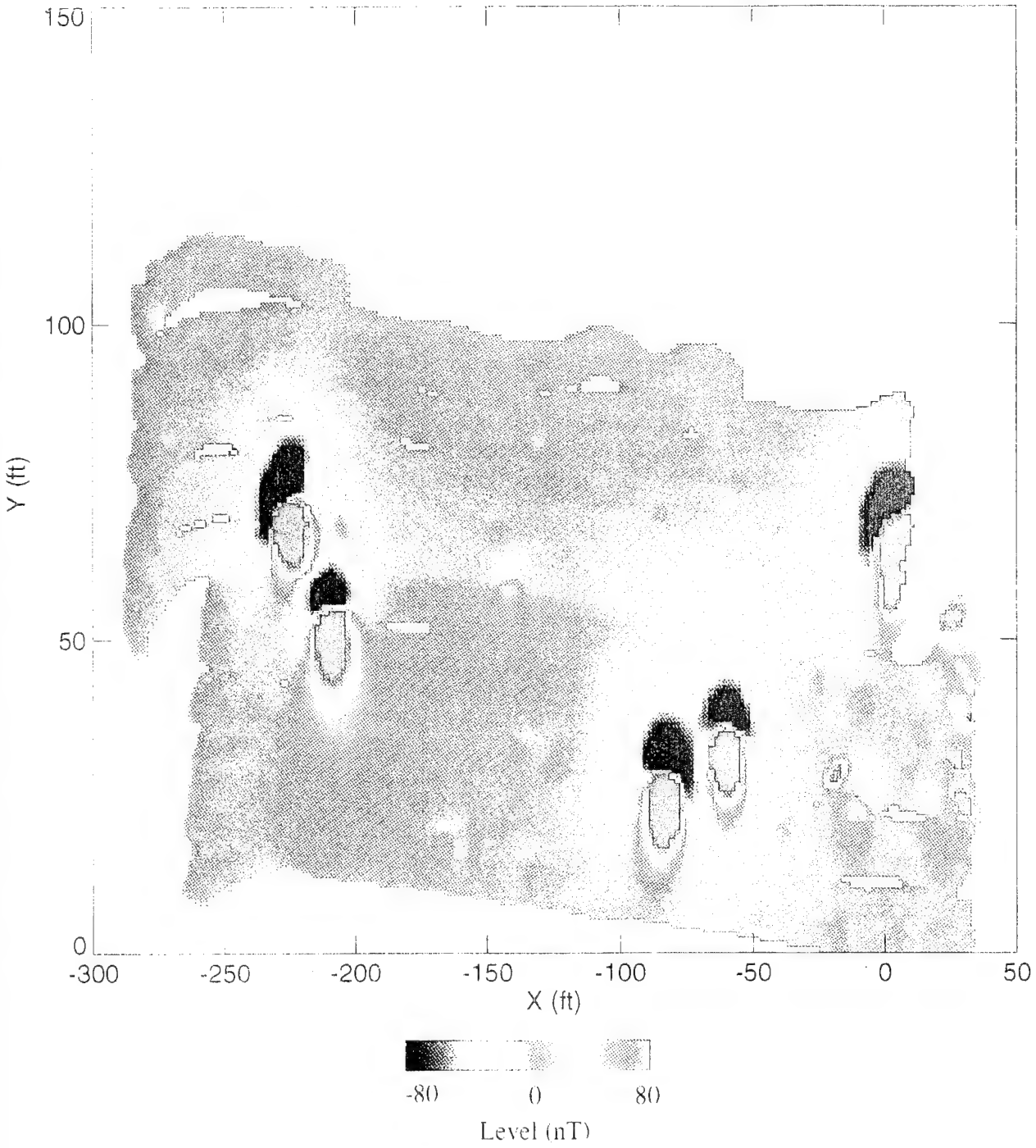


Figure 2.2 An image of SOCS array data from an unseeded field at Tyndall Air Force Base. The data have been mapped and equalized in the Magproc automatic processor.

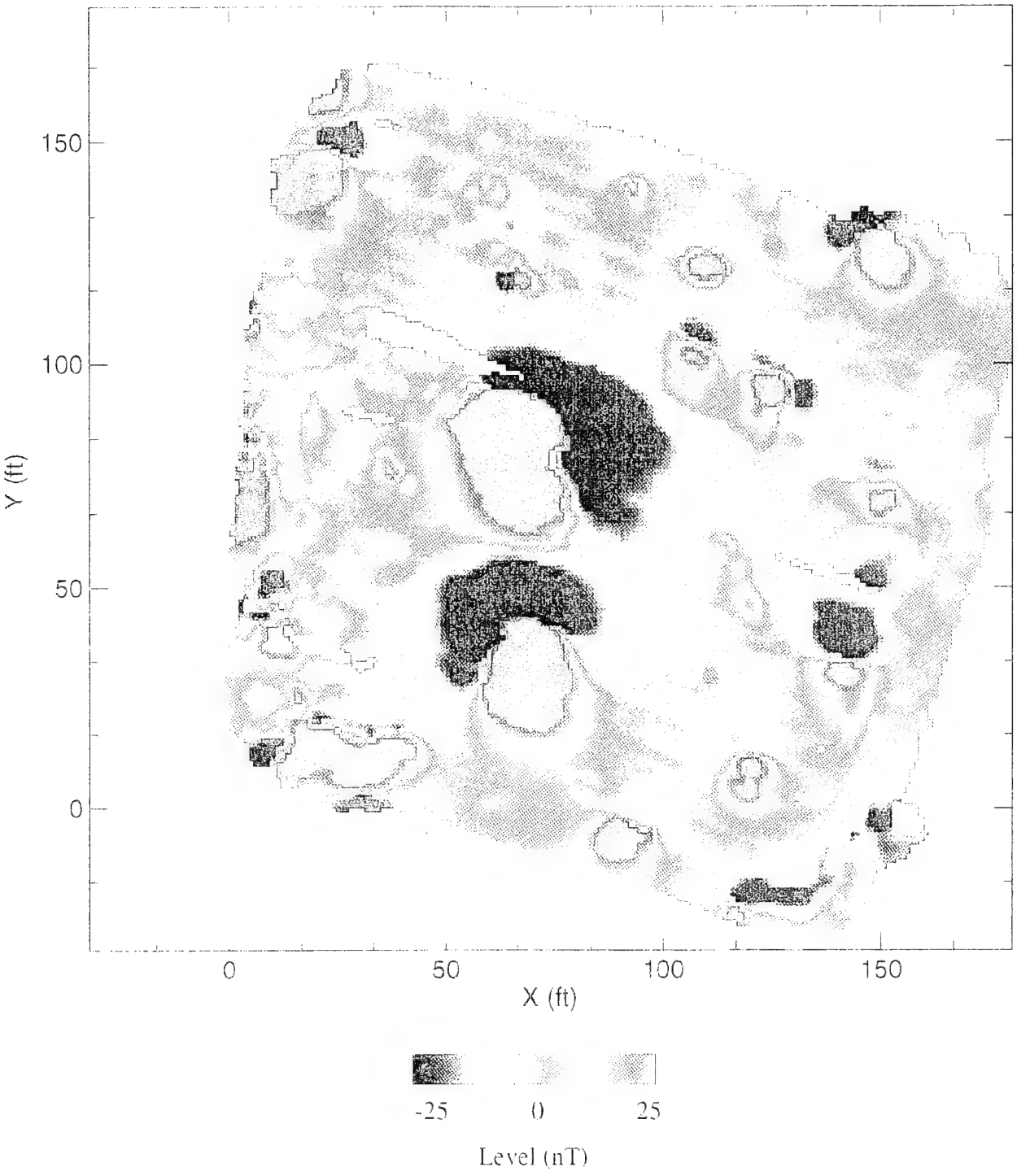


Figure 2.3 An image of SOCS array data from the seeded raised field at Tyndall Air Force Base. The data have been mapped and equalized in the Magproc automatic processor.

2.3 THRESHOLD DETECTION

The detection modules require as input a positive and negative amplitude threshold to identify target related exceedences. **Under the assumption that the underlying sensor noise field follows a gaussian distribution, either correlated or uncorrelated, then false alarm statistics can be calculated as well as target detection probabilities.** The thresholds are currently set at plus and minus four times the rms estimates provided by the *combinedata* routine. This setting is sufficiently high to suppress noise fluctuation-generated false alarms while retaining weaker targets. However, when high SNR objects are present, a low threshold will yield many points over threshold, generally over a large area. Thus, these objects, because of their large circle of influence, may in turn merge with other targets.

The thresholds are then applied to the data, and the location and amplitude of all threshold exceedences are stored. These positive and negative exceedences are separately clustered into groups of adjoining points. A logic diagram for the clustering algorithm is shown in Appendix D. Nearby positive and negative clusters are then associated to form dipole candidates. Ambiguous pairings are resolved in favor of combinations with the least distance between centers. Figure 2.4 shows the centroids of the positive and negative clusters detected in the processor for the Tyndall Air Force Base unseeded field data. Overlaid are the estimated positions of the targets after the association algorithm has been applied. Figure 2.5 shows data (run ps2) from the seeded raised field at Tyndall Air Force Base taken during the SOCS demo in April 1995. The estimated target positions have also been overlaid. In both figures target positions are initial estimates and will be used as inputs to the target parameter estimation algorithms which will obtain more precise estimates of target locations.

The primary modules in this processing area are:

A) *Exceed.c* which calculates the threshold exceedences.

B) *Cluster.c* which clusters adjoining threshold exceedences.

C) *Assoc.c* which associates plus and minus clusters to form dipole candidates.

At this point program control is passed to the target parameter estimation software.

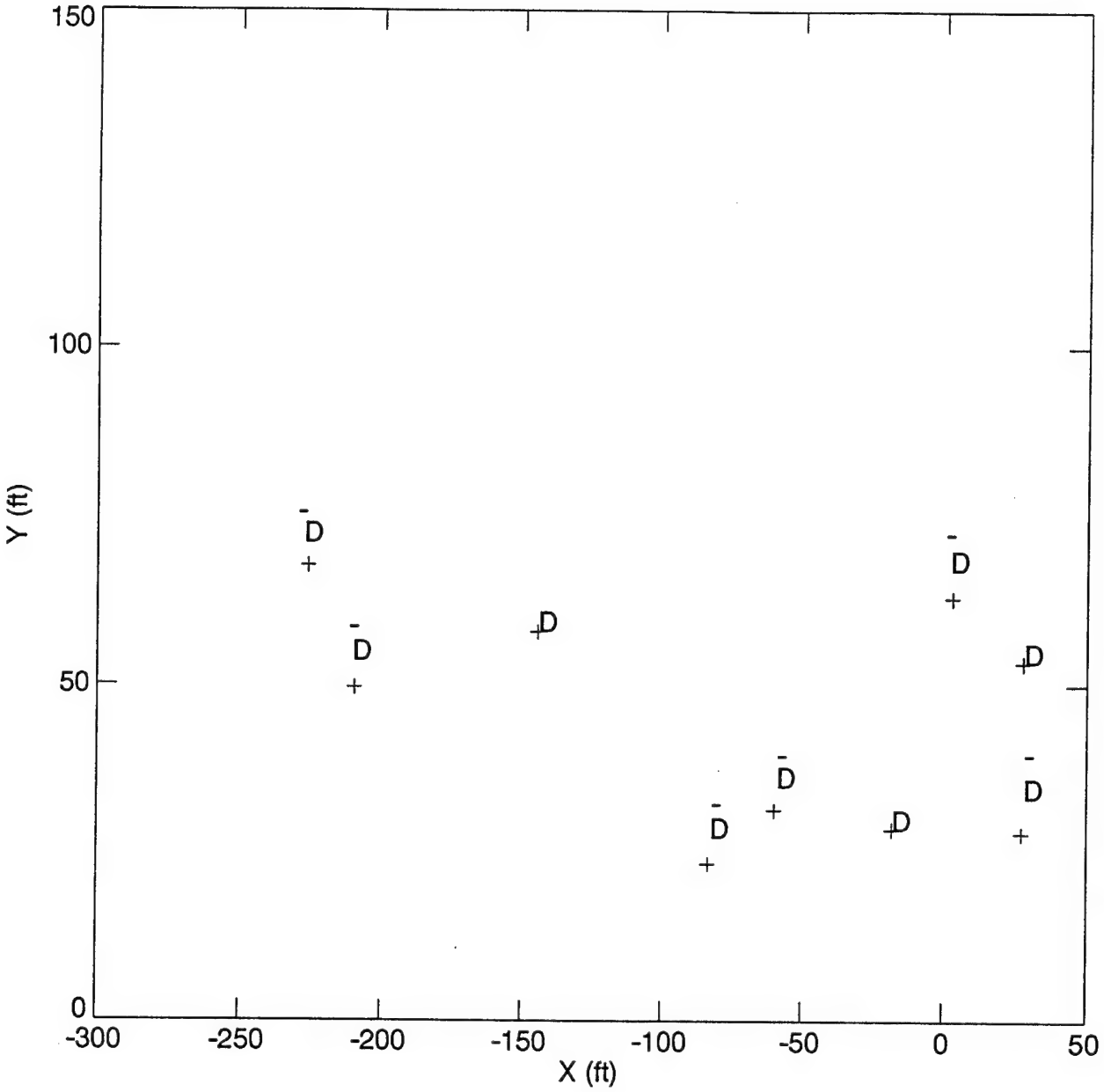


Figure 2.4 Shown are the positive and negative cluster centroid positions, denoted by "+"and "-" , output by the Magproc processor. Overlaid (denoted by "D") are the processor generated "+" and "-" cluster associations. The data are from the SOCS array and were taken at an unseeded field at Tyndall Air Force Base as shown in figure 2.2.

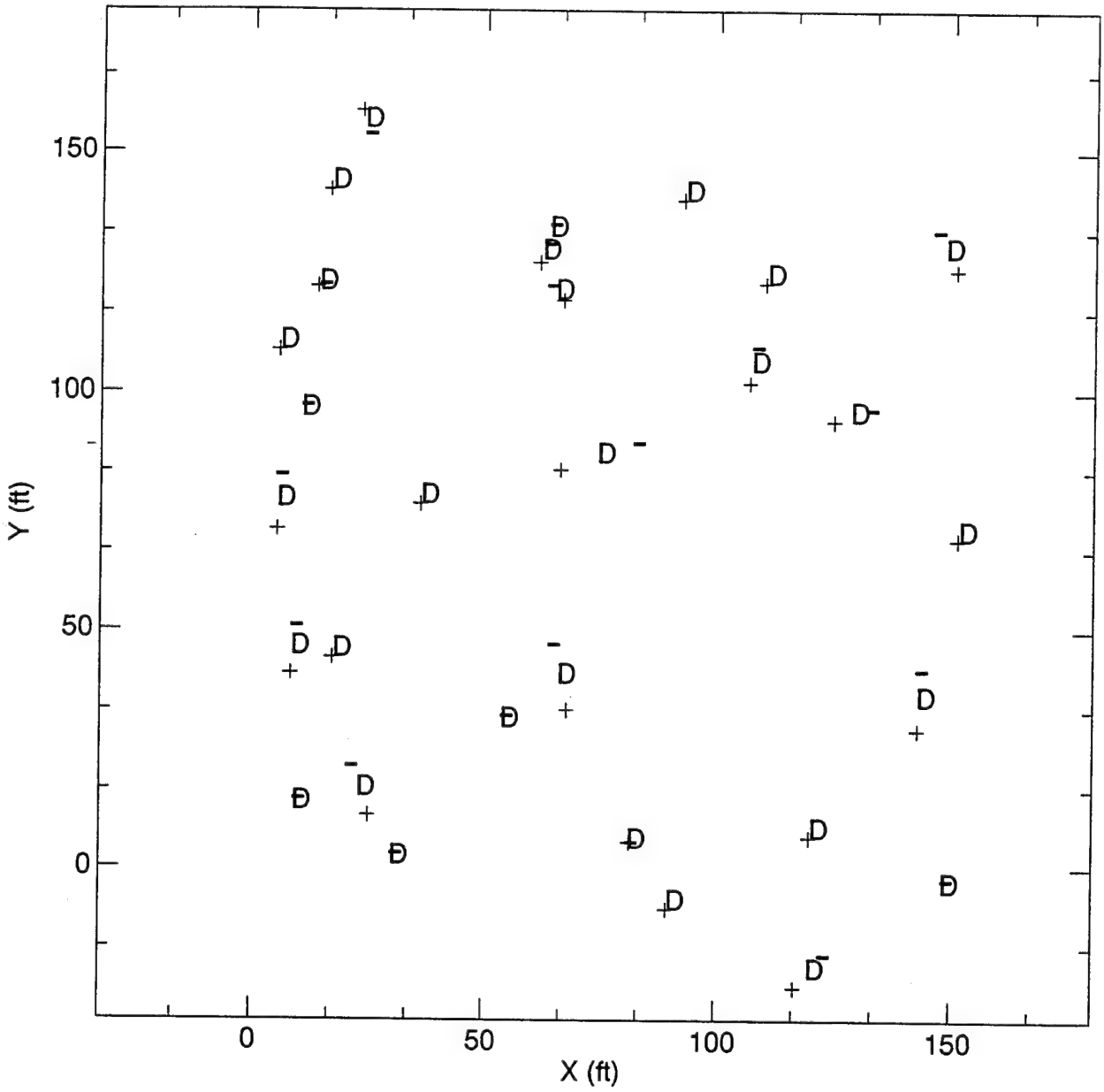


Figure 2.5 Shown are the positive and negative cluster centroid positions, denoted by "+"and "-" , output by the Magproc processor. Overlaid (denoted by "D") are the processor generated "+" and "-" cluster associations. The data are from the SOCS array and were taken at the seeded raised field at Tyndall Air Force Base as shown in figure 2.3.

2.4 TARGET PARAMETER ESTIMATION

Currently in *Magproc*, **the theoretical model used in the target parameter estimation algorithm is a magnetic dipole model.** The magnetic field anomaly due to a compact, ferrous object may always be expanded in a multi-pole expansion with the leading term representing the dipole contribution. The higher order terms fall off more rapidly with distance than the dipole term. In using the dipole model we assume that the target is compact and sufficiently far from the sensor. The dipole model is parameterized by location in three dimensions, a dipole moment, and two dipole orientation angles (the inclination and declination). The inclination and declination angles of the dipole do not necessarily reflect the orientation of the body producing the anomaly. The equation for the field produced by a magnetic dipole is given below:

$$\vec{B}_d(\vec{r}) = \frac{3\vec{n}(\vec{n}\cdot\vec{m})-\vec{m}}{|\vec{r}|^3}$$

Where m is the induced magnetic moment vector, n is a unit vector along the sensor-object line and r is the sensor-object separation vector.

Target parameter estimation begins with preliminary estimates of location (x, y, z) and orientation (inclinations and declinations) based on peak-to-peak separations of the dipole pair and their relative locations and amplitudes. Using the preliminary estimates as a starting point, **the target parameters and error bounds are estimated with a maximum likelihood fit to a dipole model.** In this processing step the amplitude is scaled out of the data so only the dipole shape is fit. The maximum likelihood search algorithm is a modified gradient search technique originally developed by Marquardt (Bevington 1969, 235-37) and contains a tuneable step-size parameter. A logic diagram for this algorithm is given in Appendix D. After the shape fit is complete, the dipole strength is determined using a second maximum likelihood fit, and this in turn is converted into an effective radius for the object. The overall quality of the fit is measured by a chi-squared/degree of freedom. Figure 2.6 shows mapped and equalized data from the

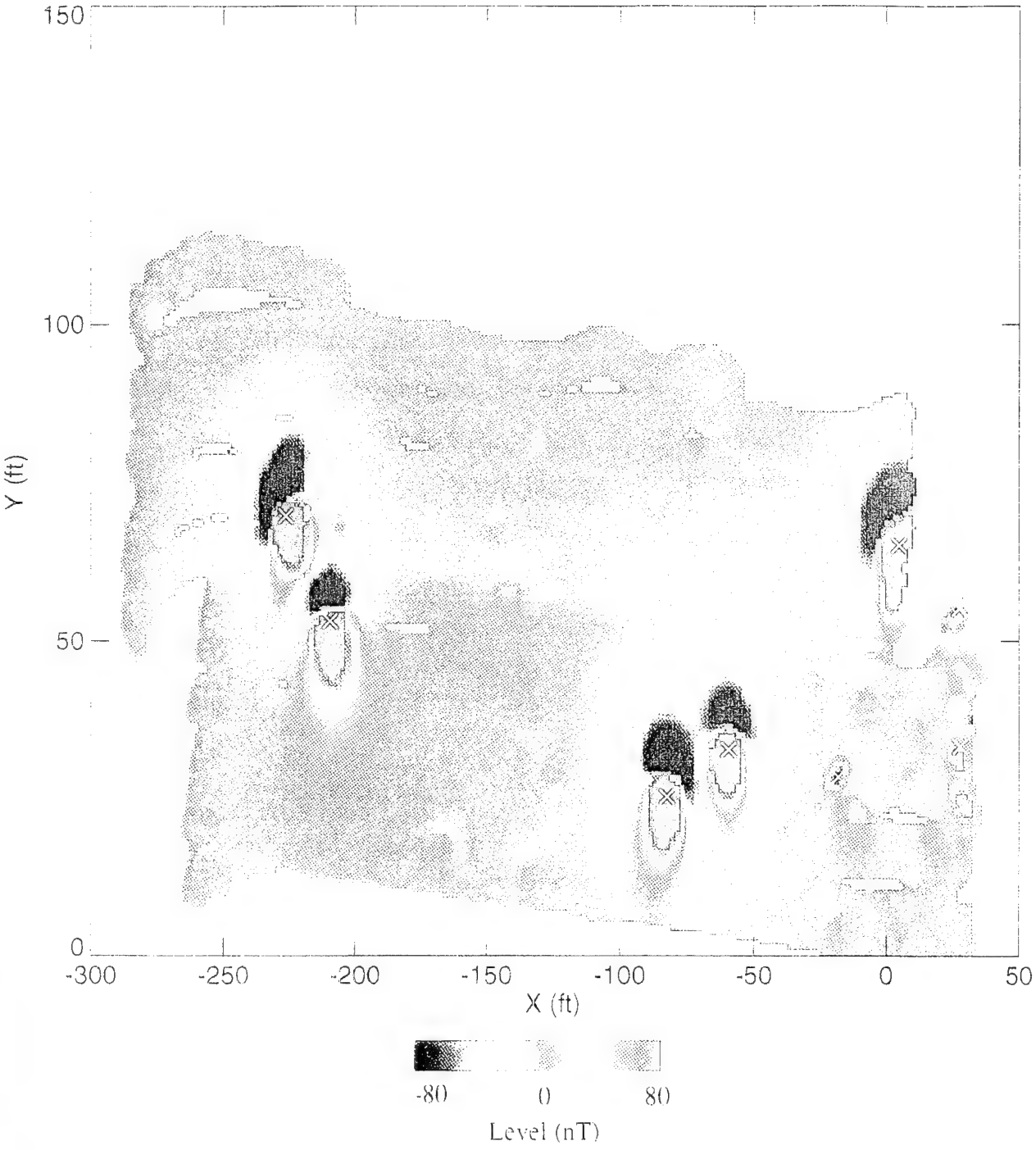


Figure 2.6 An image of SOCS array data from an unseeded field at Tyndall Air Force Base. The data have been mapped and equalized in the Magproc automatic processor. Overlaid are the fitted target positions.

unseeded field at Tyndall Air Force Base. Overlaid are the fitted object positions of targets detected by the *Magproc* processor. Figure 2.7 shows mapped and equalized data from the seeded raised field (run ps2) taken during the SOCS demo in April 1995. Overlaid are the fitted target positions which have been numbered for convenience in comparing to ground truth.

During the SOCS demo AETC was given ground truth for seven of the objects in the seeded raised field. Table 2.1 shows a comparison for the aforementioned objects of fitted lat/long to ground truth lat/long while Table 2.2 shows a comparison of fitted depth and size to ground truth depth and size. A "-" in either table indicates that the object was not detected and size indicates the radius of an equivalent induced sphere. The N.M. column corresponds to the object number in Figure 2.7.

TABLE 2.1

N.M.	TYPE	TRUE LAT (deg)	FIT LAT (deg)	TRUE LONG (deg)	FIT LONG (deg)
21	81mm	29.969913	29.969915	85.476160	85.476168
13	2000 lb	29.969764	29.969772	85.476223	85.476231
15	8 inch	29.969792	29.969789	85.476059	85.476064
10	BDU33	29.969610	29.969598	85.476022	85.476013
-	60mm	29.969528	-	85.476283	-
-	BDU33	29.969659	-	85.475982	-
-	RUSS MORT	29.969773	-	85.475992	-

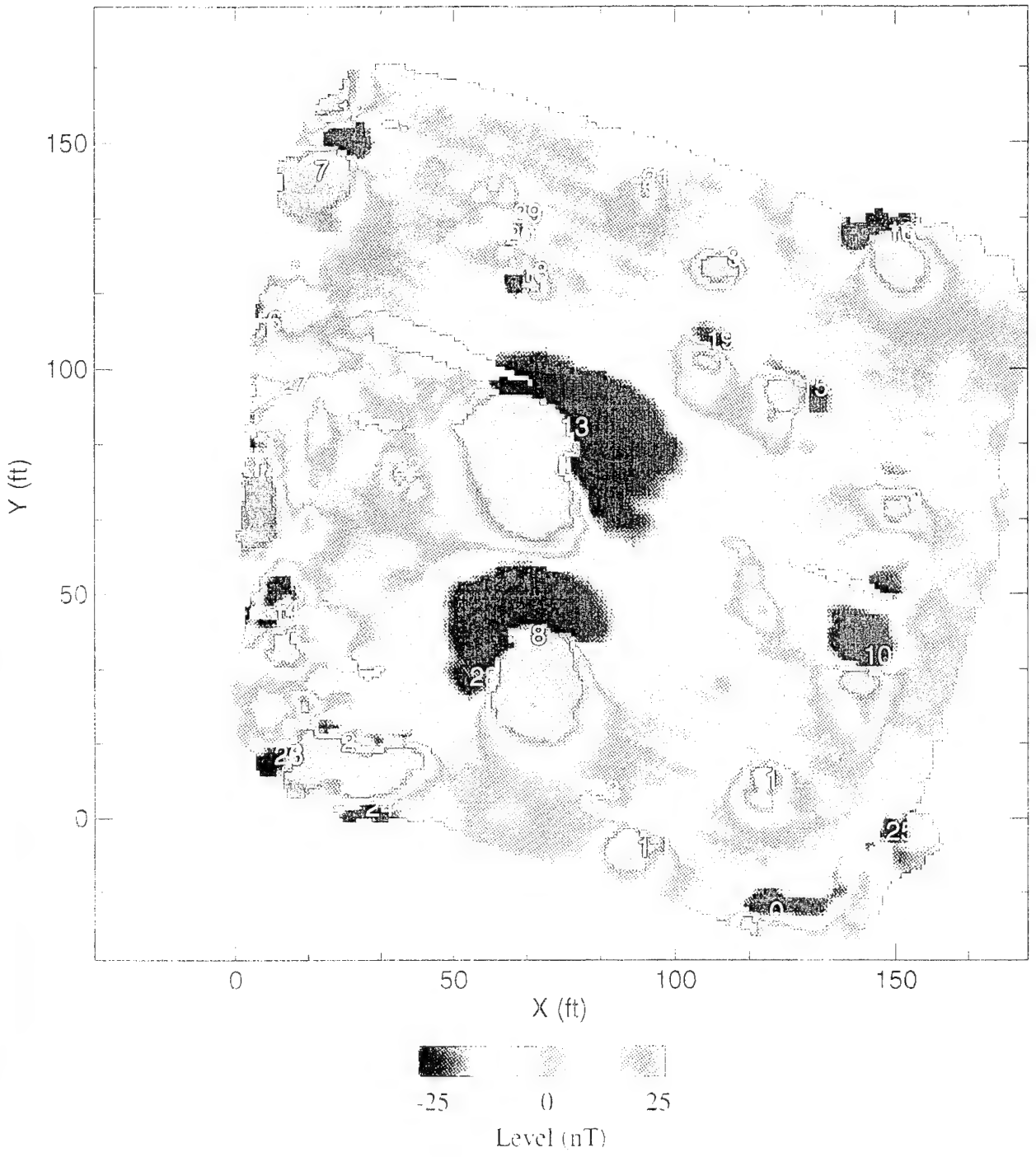


Figure 2.7 An image of SOCS array data from the seeded raised field at Tyndall Air Force Base. The data have been mapped and equalized in the Magproc automatic processor. Overlaid are the fitted target positions.

TABLE 2.2

N.M.	TYPE	TRUE DIA (m)	FIT DIA (m)	TRUE DEPTH (m)	FIT DEPTH (m)
21	81mm	0.17	0.16	-0.57	-0.43
13	2000 lb	0.83	1.14	-1.40	-2.30
15	8 inch	0.36	0.30	-0.48	-0.46
10	BDU33	0.21	0.60	-0.96	-2.20
-	60mm	0.10	-	-0.41	-
-	BDU33	0.21	-	-1.20	-
-	RUSS MORT	0.23	-	-1.50	-

Overall, **four out of the seven objects were detected by the** *Magproc* **processor**. The other three objects had peak signals below the detection threshold (plus or minus 25 nT) set by *Magproc* during this run. Of the four objects detected, objects 21 and 15 gave very good size-depth estimates while objects 13 and 10 gave poorer size-depth estimates. For the case of object 13, a shallow 2000 lb bomb we attribute this to the inadequacy of a point-dipole model to represent a shallow, extended object. The poor fit for object 10 was attributed to misassociation with a nearby unidentified object. Of the four detected UXO, objects 21, 15 and 13 were within a meter of their expected positions while object 10 was approximately 1.5 meters from its expected position.

In certain circumstances a dipole fit to a given object may not be possible. In such case, the algorithm may become unstable or may give unphysical results. The main fitting module then returns a flag which, if non-zero, indicates an unsuccessful fit. However, the x and y locations based on the initial target parameter estimates are still recorded in the output STD file.

The primary modules in the processor target characterization area are:

1) *Initparams.c* which estimates the initial dipole shape parameters.

2) *Expand.c* which expands the target candidate's circle of influence if not enough points are available for the fit.

3) *Dipole.c* which calls two main routines, *curfit.c* and *ampfit.c*, which perform the maximum likelihood shape and dipole strength fits, respectively.

After an attempt is made to fit an object to a dipole model, program control is passed to the output module.

2.5 OUTPUT

This is the final module in the processor chain and consists of a single subroutine *outstd.c*. This routine outputs data to a file in standard STD format and branches on whether or not a successful fit is achieved in the target characterization code area. If a successful fit is achieved, the target location, depth, size (S, M or L), and size confidence are reported. If, on the other hand, a fit was unsuccessful then only the location of the target is reported. This location is estimated in the *initparams.c* routine before fitting has begun. Data fields that are not estimated are filled with an asterisk. Appendix C shows a sample STD file for data (run ps2) taken at the seeded raised field at Tyndall Air Force Base during the SOCS demo in April 1995. Upon completion of the output of the STD file a *Magproc* run terminates.

This page is intentionally left blank

3.0 PROCESSING UPGRADES

In the following sections, processor upgrades are considered from the point of view of an implementation time frame. We first discuss upgrade options that could be implemented in a four to eight week period. After this, we address upgrade options which would require a more significant development time.

3.1 SHORT TERM UPGRADES

We have identified three short term upgrade options for version 1.0 of *Magproc*. They are:

A) Temporal and spatial long scale noise removal to increase detection performance.

B) Adding IDS I/O capability which would give the processor the flexibility to handle a variable number of sensors.

C) Adding a graphical interface capability with Precision Visuals PV-wave software product for target and data quality displays.

The above options are reviewed in order.

3.1.1 Long Scale Spatial and Temporal Noise Removal

Long scale temporal noise can be removed through the utilization of the reference magnetometer. When the reference channel is present, the noise subtraction module will use data from this channel to coherently subtract out the recorded noise. Experience with total field magnetometers in the past has indicated that large scale changes in the total field intensity are at lower frequency than the actual signals from ferrous objects, and come from sources such as magnetic storms. Thus, these changes can be removed without affecting target signals.

Similar to long scale temporal noise, total field magnetometers detect large spatial scale environmental noise. This noise source typically comes from local geology and generally has

spatial scale lengths that are larger than UXO targets of interest. Thus, contributions from these sources can be filtered out in the noise module through high-pass spatial filtering. Implementation of this module will significantly drop detection thresholds and consequently, will enable the detection of weaker targets. However, this technique will not improve detection thresholds for impact ranges with significant amounts of surface clutter since these objects occupy shorter spatial scales and generally overlap the phase space of UXO targets of interest.

3.1.2 IDS I/O Capability

At present, the processor does not accept IDS type data structures which are header files that contain information about the remediation site, and the types and number of instruments used. In addition information such as the tow bar length and the positions of the mags on the cart can also be carried in IDS structures. Version 1.0 of *Magproc* assumes a fixed number of sensors, sensor heights, data rates and mag positions on the cart. We propose to add I/O modules to both input and output IDS data structures and to modify *Magproc* to allow for a variable number of sensors and other IDS carried header variables.

3.1.3 Graphical Interface Capability

AETC has much experience with a powerful analysis and graphical interactive language called PV-wave. It is relatively straightforward to couple *Magproc* to PV-wave, giving the user the ability to view raw data, intermediate processed data, or final target detections and fit results.

3.2 LONG TERM UPGRADES

We have identified two major long term upgrades for version 1.0 of *Magproc*. They are:

A) Spatial optimal matched filtering modules to increase detection performance.

B) Non-dipole parameter estimation modules to classify objects which are poorly represented by the dipole model.

3.2.1 Optimal Matched Filtering

The technique of optimal matched filtering involves the coherent correlation of measured sensor data with model signal templates called matched filters. The correlation is whitened by the power spectral density estimate of the local noise field and is maximal in the presence of gaussian noise.

Optimal matched filter processing affords several advantages over threshold detection, currently used in *Magproc*:

A) While threshold detection treats data points independently, optimal matched filtering exploits the spatial coherence of the signal by performing a simultaneous correlation against all signal points.

B) In the presence of red noise, threshold detections will be driven by longer length scales, which have a higher rms level. Optimal matched filtering naturally accommodates red noise through pre-whitening by the noise power spectral density.

C) In the clustering algorithm high SNR targets, due to a large circle of influence, will tend to merge with other nearby clusters. However, with optimal matched filtering this problem will be avoided.

Implementation of matched filtering modules would significantly improve the detection performance of the processor and replace existing detection modules.

3.2.2 Non-dipole Characterization Modules

In many circumstances in total field magnetometer data, the dipole model may not be a good representation of the data. For example, high aspect ratio targets such as bombs, pipes, and drums, depending on their burial depth and degree of elongation, may or may not look like a dipole, and the dipole model may or may not yield reasonable estimates of size and depth. We

have found that a finite/infinite cylinder model is appropriate for targets such as pipes and shallow drums, while a prolate/oblate spheroid model can be used to address long shells and bombs. Besides fitting the parameters of location and depth, each of these models would provide information about orientation and relative dimensions (aspect ratio). Implementation of these non-dipole modules would give the processor more flexibility in its characterization of detected targets.

4.0 REFERENCES

1) SOCS Demonstration at Jefferson Proving Ground: Quick Look Data Summary

2) Bevington, Philip R. 1969. Data Reduction and Error Analysis for the Physical Sciences. New York: McGraw-Hill Book Company.

This page is intentionally left blank

APPENDIX A

INSTRUCTIONS FOR COMPILING AND LINKING MAGPROC ON A SUN SPARC10

The user will receive a diskette which contains two tar files: A) *PROJ.tar*, which contains the code necessary to create a cartographic library. This library is necessary for lat/long to UTM xy conversions and vice-versa and will be linked to Magproc source code after its compilation. B) *Magproc.tar*, which contains source code necessary to create the Magproc processor.

Below are step by step instructions for compilation and installation of Magproc.

STEP 1: Create a directory to contain the aforementioned tar files. This directory will contain the processor.
STEP 2: Copy the two tar files to this directory.
STEP 3: *tar xvf PROJ.tar* to extract cartographic lib files from archives.
STEP 4: *cd PROJ.4* to step down one level.
STEP 5: *./configure --prefix=/your/path* which will compile the cartographic code. It will also create a subdirectory called lib under */your/path*.
STEP 6: *sh install* to install the cartographic library.
STEP 7: *cd..* to move up one level.
STEP 8: *tar xvf Magproc.tar* to extract Magproc source files.
STEP 9: *make Magproc* to compile and link.
STEP 10: *Magproc <CR>* to execute processor.

APPENDIX B

MAGPROC SOURCE CODE LISTING

```
/*****************************************************************************
*
*     Routine: ampfit.c
*
*     This module fits a dipole amplitude(shape has already been calculated)
*     from Bevington's Data Reduction and Error Analysis(pgs 213).
*
*     Doug DeProspo
*     17Nov94
*     10Jul95 : Added earth mag field params to argument list.
*
*              Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include "dipole.h"

void ampfit(int npts,double *xf,double *yf,double *af,double param[],double sigma,
            double efield[],double ampstep,double *amp,double *amperr,double *chisq)


{

    double free,fn,diff,chisq1,chisq2,chisq3,save,weight ;
    double chitest,amptest,amperrtest ;
    double x0,y0,z0,inc,dec ;
    double *b0 ;
    int chicheck = 1 ;
    int nfree ;


/*  start main body  */

    b0 = dvector(0,npts) ;

    nfree = npts-1 ;
    free = (double)nfree ;
    weight = 1./(sigma*sigma) ;
    amptest = *amp ;

/*  Start using fitted shape parameters from curfit and initial amp estimate */

    getpar(param,&x0,&y0,&z0,&inc,&dec) ;
    btotal2(x0,y0,z0,amptest,inc,dec,xf,yf,efield,b0,npts) ;
    chisq1 = chi(af,weight,npts,nfree,b0) ;
    fn = 0. ;

    amptest = amptest + ampstep ;
    btotal2(x0,y0,z0,amptest,inc,dec,xf,yf,efield,b0,npts) ;
    chisq2 = chi(af,weight,npts,nfree,b0) ;

    diff = chisq1-chisq2 ;
```

```
    if (diff < 0)
      {
      ampstep = -1.*ampstep ;
      amptest = amptest + ampstep ;
      btotal2(x0,y0,z0,amptest,inc,dec,xf,yf,efield,b0,npts) ;
      save = chisq1 ;
      chisq1 = chisq2 ;
      chisq2 = save ;
      }

      while (chicheck == 1)
          {
       fn = fn +1. ;
       amptest = amptest + ampstep ;
       btotal2(x0,y0,z0,amptest,inc,dec,xf,yf,efield,b0,npts) ;
       chisq3 = chi(af,weight,npts,nfree,b0) ;
       diff = chisq3 - chisq2 ;
       if (diff < 0)
         {
         chisq1 = chisq2 ;
         chisq2 = chisq3 ;
         }
       else
         {
         chicheck = 0 ;
         }
       }
    ampstep = ampstep* (1./(1. + (chisq1-chisq2)/(chisq3-chisq2)) + .5) ;
    amptest = amptest - ampstep ;
    amperrtest = fabs(ampstep * sqrt(2./(free*(chisq3-2.*chisq2+chisq1)))) ;

    btotal2(x0,y0,z0,amptest,inc,dec,xf,yf,efield,b0,npts) ;
    chitest = chi(af,weight,npts,nfree,b0) ;

    *amp = fabs(amptest) ;
    *amperr = amperrtest ;
    *chisq = chitest ;

    free_dvector(b0,0,npts) ;
    return ;

}
```

```
/*****************************************************************************
*
*     Routine: assoc.c
*
*     This module takes the positive and negative clusters
*     and associates them to make dipoles.
*
*     Doug DeProspo
*     21Dec94
*     11Jul95 : Modified to fix ++- ambiguity resolution bug in al
*
*          Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include "dipole.h"


void assoc(double *cenxp,double *cenyp,double *cenxm,double *cenym,
        int nclusterp,int nclusterm,int *ndipole,int *indexp,int *indexm)



{

    int *ci,*cj,**index,**index2,
        *flag ;
    int i,j,jmin,k,l,kother,kmin,testgood,testbad ;
    int mdipole=0,neg ;
    double **distp,**distm ;
    double dist,deltax,deltay,distmin ;

    ci = ivector(0,maxdipoles) ;
    ivector_init(ci,maxdipoles,0) ;

    cj = ivector(0,maxdipoles) ;
    ivector_init(cj,maxdipoles,0) ;

    index = imatrix(0,nclusterp,0,maxdipoles) ;
    imatrix_init(index,nclusterp,maxdipoles,0) ;

    index2 = imatrix(0,nclusterm,0,maxdipoles) ;
    imatrix_init(index2,nclusterm,maxdipoles,0) ;

    distp = dmatrix(0,nclusterp,0,maxdipoles) ;
    dmatrix_init(distp,nclusterp,maxdipoles,9999.) ;

    distm = dmatrix(0,nclusterm,0,maxdipoles) ;
    dmatrix_init(distm,nclusterm,maxdipoles,9999.) ;


/*  Find all +/- pairings within cut2 including ambiguities */
```

```
for (i=0; i < nclusterp ;i++)
  {
  for (j=0; j < nclusterm ;j++)
       {
    deltax = cenxp[i]-cenxm[j] ;
    deltay = cenyp[i]-cenym[j] ;
    dist = pow(deltax*deltax+deltay*deltay,.5) ;
    if (dist < cut2)
      {
      index[i][ci[i]] = j ;
      index2[j][cj[j]] = i ;
      distp[i][ci[i]] = dist ;
      distm[j][cj[j]] = dist ;
      ci[i] += 1 ;
      cj[j] += 1 ;
      }

    }
  }

/*   Resolve ambiguous associations and account for unassociated + peaks */

  flag = ivector(0,nclusterm) ;
  ivector_init(flag,nclusterm,0) ;

  for (i=0; i < nclusterp;i++)
       {
    if (ci[i] > 0)
/*  We have an associated + peak */
       {
      if (ci[i] == 1)
/*  We have a single - associated to this + */
        {
        if (flag[index[i][0]] == 0)
/*  This - was not associated to a previous + */
          {
          indexp[mdipole] = i ;
          indexm[mdipole] = index[i][0] ;
          mdipole += 1 ;
          flag[index[i][0]] = 1 ;
          }
        else
/*  This - was associated to a previous + */
          {
          neg = index[i][0] ;
          distmin = 9999. ;
          for (k = 0 ; k <= cj[neg] -1 ; k++)
               {
           if (distp[neg][k] < distmin)
             {
             distmin = distp[neg][k] ;
```

```
                kmin = k ;
                }
              }
             for (l=0 ; l <= 1 ; l++) /* assume 2-fold ambiguity */
              {
              if ( l != kmin){kother = l ;}
              }
             testgood = index2[neg][kmin] ;
             testbad = index2[neg][kother] ;
             if (testgood == i) /* current + is proper choice */
              {
              indexp[mdipole] = i ;
              indexm[mdipole] = neg ;

/*   Now break link from earlier misassociation */
              for (l=0 ; l <= mdipole-1 ; l++)
                    {
                if (indexp[l] == testbad){indexm[l] == -1 ; }
                }
              mdipole += 1 ;
              }
            else
              {
              indexp[mdipole] = i ;
              indexm[mdipole] = -1 ; /* default for single + peak */
              mdipole += 1 ;
              }
            }
          }
        else
/*   We have more than one - associated to this + */
          {
          jmin = -1 ;
          distmin = 9999. ;
          for (j=0; j < ci[i];j++)
                {
/*   Test if current - was previously associated to another + */
            if (flag[index[i][j]] != 1)
              {
              if (distp[i][j] < distmin)
/*   Resolve ambiguity by closest distance */
                    {
                distmin = distp[i][j] ;
                jmin = j ;
                    }
              }
            }
        if (jmin == -1)
/*   All - candidates were previously associated to other + */
          {
          indexp[mdipole] = i ;
```

```
                indexm[mdipole] = -1 ; /* default for single + peak */
                mdipole += 1 ;
                }
              else
                {
                indexp[mdipole] = i ;
                indexm[mdipole] = index[i][jmin] ;
                flag[index[i][jmin]] = 1 ;
                mdipole += 1 ;
                for (j=0; j < ci[i];j++)
                      {
                  if (j != jmin)
 /*   Remove this association from the - that weren't chosen */
                    {
                    cj[index[i][j]] -= 1 ;
                    }
                  }
                  }
             }
           }
         else
 /*   We have an unassociated + peak */
           {
           indexp[mdipole] = i ;
           indexm[mdipole] = -1 ; /* default for single + peak */
           mdipole += 1 ;
           }
        }

 /*   Loop over - clusters to see if any were unassociated */

      for (j=0; j < nclusterm;j++)
          {
        if (cj[j] == 0)
          {
          indexp[mdipole] = -1 ; /* default for single - peak */
          indexm[mdipole] = j ;
          mdipole += 1 ;
          }
        }

      *ndipole = mdipole ;

      free_ivector(flag,0,nclusterm) ;
      free_ivector(ci,0,maxdipoles) ;
      free_ivector(cj,0,maxdipoles) ;
      free_imatrix(index,0,nclusterp,0,maxdipoles) ;
      free_dmatrix(distp,0,nclusterp,0,maxdipoles) ;

      return ;
}
```

```
/*****************************************************************************
 *
 *    Routine: btotal2.c
 *
 *    This module is passed a vector of points and based on a magnetic
 *    dipole model returns the total field at these points.
 *
 *    Doug DeProspo
 *    27Oct94
 *    10Jul95 : Added earth mag field params to argument list.
 *
 *          Arete Engineering Technologies Corporation
 *****************************************************************************
 */
#include "dipole.h"

#define pi 3.14159
#define dtr pi/180.
#define BE 50000.



void btotal2(double x0,double y0,double z0,double a0,double inc,double dec,
             double *x,double *y,double efield[],double *btot,int npoints)
{

    double m,mx,my,mz,r,rx,ry,rz ;
    double rdm,bxd,byd,bzd ;
    double IncE,DecE,BxE,ByE,BzE ;
    int i ;

/*  start main body  */


    IncE = efield[0] ;
    DecE = efield[1] ;
    IncE *= dtr ;
    DecE *= dtr ;

/* Earth vector components */

    BxE = BE*cos(IncE)*cos(DecE) ;
    ByE = BE*cos(IncE)*sin(DecE) ;
    BzE = BE*sin(IncE) ;

/*  Induced Moment  */
    m = BE*pow(a0,3.)  ;
    mx = cos(inc*dtr)*cos(dec*dtr)  ;
    my = cos(inc*dtr)*sin(dec*dtr)  ;
    mz = sin(inc*dtr) ;
```

```
    for (i=0; i < npoints; i++)
      {
       rx = x[i]-x0 ;
       ry = y[i]-y0 ;
       r = sqrt(rx*rx + ry*ry + z0*z0) ;
       rx = rx/r ;
       ry = ry/r ;
       rz = -z0/r ;
       rdm = rx*mx + ry*my + rz*mz ;

/*     Dipole Field  */
       bxd = (m/pow(r,3.))*(3.*rdm*rx-mx) ;
       byd = (m/pow(r,3.))*(3.*rdm*ry-my) ;
       bzd = (m/pow(r,3.))*(3.*rdm*rz-mz) ;

/*     Total Field  */
       btot[i] = sqrt(BE*BE + 2.*(bxd*BxE + byd*ByE +bzd*BzE) +
                      (bxd*bxd + byd*byd + bzd*bzd)) - BE ;
      }
      return ;
}
```

```
/*****************************************************************************
 *
 *    Routine: centroid.c
 *
 *    This module takes a group of clusters and finds
 *    the x,y centroid of each cluster
 *
 *    Doug DeProspo
 *    21Dec94
 *
 *          Arete Engineering Technologies Corporation
 *****************************************************************************
 */
#include "dipole.h"


void centroid(int ncluster,int *count,double **xcluster,double **ycluster,
        double **acluster,double *cenx,double *ceny)


{

    int i,j ;
    double total ;

    for (i=0; i < ncluster;i++)
        {
      total = 0. ;
      for (j=0; j < count[i]-1; j++)
          {
       total += acluster[i][j] ;
       }
      for (j=0; j < count[i]-1; j++)
          {
       cenx[i] += (acluster[i][j]*xcluster[i][j])/total ;
       ceny[i] += (acluster[i][j]*ycluster[i][j])/total ;
       }
     }

   return ;

}
```

```
/*****************************************************************************
*
*     Routine: checkparam.c
*
*     This module checks the fitted (x,y,z) against the initial
*     x,y,z estimates. If too much change has occurred then the
*     fit is declared invalid(eflag=1).
*
*
*     Doug DeProspo
*     31Jan95
*
*                Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include "dipole.h"

void  checkparam(double param[],double initparam[],int *eflag)


{

    double dx,dy,dz ;
    int flag = 0 ;

/*  start main body  */


    dx = fabs(param[0]-initparam[0]) ;
    dy = fabs(param[1]-initparam[1]) ;
    dz = fabs(param[2]-initparam[2]) ;
    if ((dx > 20.) ||  (dy > 20) || (dz >20.) || (param[2] < 0.) || (param[2] > 30.))
      {
      *eflag = 1 ;
      return ;
      }
    *eflag = flag ;
    return ;
}
```

```
/*****************************************************************************
*
*    Routine: cluster.c
*
*    This module takes exceedence input x,y,amp vectors
*    and creates output cluster groupings
*
*    Doug DeProspo
*    21Dec94
*
*            Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include "dipole.h"


void cluster(int nexceed,double *xexceed,double *yexceed,double *aexceed,int *ncluster,
          int *count,double **xcluster,double **ycluster,double **acluster)


{

    int icl = 0,mcluster = 0 ;
    int *ifl ;
    int i,j,k ;
    double deltax,deltay,dist ;

    ifl = ivector(0,nexceed) ;
    ivector_init(ifl,nexceed,0) ;

    for (i=0; i< nexceed-1;i++)
      {
     if (ifl[i] == 0)
       {
       icl += 1 ;
       ifl[i] = icl ;
       }
     for (j=0; j < nexceed;j++)
         {
       if (ifl[i] != ifl[j] )
         {
         deltax = xexceed[i]-xexceed[j] ;
         deltay = yexceed[i]-yexceed[j] ;
         dist = pow((deltax*deltax+deltay*deltay),.5) ;
          if (dist < cut)
            {
           if (ifl[j] == 0)
             {
             ifl[j] = ifl[i] ;
             }
           else
```

```
            {
            for (k=0; k < nexceed;k++)
                  {
              if (ifl[j] == ifl[k])
                {
                ifl[k] = ifl[i] ;
                }
              }
            }
          }
        }
      }
    }

/*  Do cleanup of ifl array and icl counter */

    for (i=0; i < nexceed;i++)
      {
     count[ifl[i]] += 1 ;
      }

    for (i=1; i <= icl;i++)
      {
      if (count[i] != 0)
        {
        mcluster += 1 ;
        count[mcluster-1] = count[i] ;
        for (j=0; j < nexceed; j++)
            {
          if (ifl[j] == i)
            {
            ifl[j] = mcluster-1 ;
            }
          }

      }
     }

    for (i=0; i < mcluster; i++)
        {
     count[i] = 0 ;
     }
    for (i=0; i < nexceed;i++)
        {
     xcluster[ifl[i]][count[ifl[i]]] = xexceed[i] ;
     ycluster[ifl[i]][count[ifl[i]]] = yexceed[i] ;
     acluster[ifl[i]][count[ifl[i]]] = aexceed[i] ;
     count[ifl[i]] += 1 ;
     }
```

```
    *ncluster = mcluster ;

    free_ivector(ifl,0,nexceed) ;

    return ;

}
```

```
/*****************************************************************************
*
*    Routine: combinedata.c
*
*    This routine take out the median for the good sensors
*    in both directions and combines x,y,amp from 4 sensors
*    into a single x,single y and single amp array. Also returns
*    the overall field rms level for the n good sensors
*
*
*
*    Doug DeProspo
*    27Jun95
*              Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include "dipole.h"


void combinedata(Map *pmap,long ngoodmap,int insflag[],int *dirflag
              ,double insmedian[],double insrms[],double *xdata,
              double *ydata,double *ampdata,long *ncount,double *sigma)


{

    long mcount=0,i,medindex,lowindex,highindex ;
    double *temp,sumrms ;
    int j,goodcount ;



    for (i=0 ; i < ngoodmap; i++)
     {
     for (j=0 ; j <=3 ; j++)
       {
       if (insflag[j] != 1)
         {
         if (dirflag[i] == 1)
           {
           xdata[mcount] = pmap->xmap[j][i] ;
           ydata[mcount] = pmap->ymap[j][i] ;
           ampdata[mcount] = (pmap->amp[j][i]) - insmedian[j] ;
           mcount += 1 ;
           }
         else
           {
           xdata[mcount] = pmap->xmap[j][i] ;
           ydata[mcount] = pmap->ymap[j][i] ;
           ampdata[mcount] = (pmap->amp[j][i]) - insmedian[j+4] ;
           mcount += 1 ;
```

```
            }
          }
        }
      }

/*   Now determine the overall rms for the combined amp fields */

       sumrms = 0. ;
       goodcount = 0 ;
     for (j=0 ; j<8 ; j++)
       {
       if (insflag[j] != 1)
         {
         goodcount += 1 ;
         sumrms += insrms[j] ;
         }
       }
     *sigma = sumrms / goodcount ;
     *ncount = mcount ;

     free_dvector(temp,0,mcount) ;

     return ;

}
```

```
/*****************************************************************************
*
*     Routine: conmin.c
*
*     This routine converts lat or long in degrees and fractional degrees
*     to degrees,minutes and fractions of minutes.
*
*     Doug DeProspo
*     30Jun95
*          Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include "dipole.h"


void conmin(double l,double *ldeg,double *lmin)

{

    double fracdeg,temp,deg,min,fracmin ;


    fracdeg = modf(l,&deg) ;

    fracmin = fracdeg * 60. ; /* gets minutes and fractional minutes */



    *ldeg = deg ;
    *lmin = fracmin ;


    return ;
}
```

```
/*****************************************************************
****
*
*      Routine: contoll.c
*
*      This module converts utm xy to lat/long. The
*      reference is the zone which is passed as an
*      argument.
*
*      Doug DeProspo
*      29Jun95
*      10Jul95 : Modified  to allow non-default geoid params.
*
*          Arete Engineering Technologies Corporation
*****************************************************************/
#include <stdio.h>
#include <projects.h>


void contoll(double x,double y,short zone,double *lat,double *lon)



{
   static char *parms[5] ;
   char source1[] = "proj=utm" ;
   char source2[] = "          " ;
   char source3[] = "inv" ;
  char source4[] = "ellps=WGS84" ;
  char source5[] = "no_defs" ;


   PJ *ref ;
   UV data ;


  parms[0] = source1 ;
  sprintf(source2,"zone=%hd",zone) ;
  parms[1] = source2 ;
  parms[2] = source3 ;
  parms[3] = source4 ;
  parms[4] = source5 ;


  if (! (ref = pj_init(5,parms)))
  {
  fprintf(stderr,"Projection Initialization Failed\n") ;
  exit(1) ;
  }

  data.u = x ;
```

```
    data.v = y ;
    data = pj_inv(data,ref) ;
    *lat = data.v / DEG_TO_RAD ;
    *lon = data.u / DEG_TO_RAD ;

    return ;
}
```

```
/*****************************************************************
****
*
*     Routine: contoxy.c
*
*     This module converts lat/long to utm xy. The
*     reference is the zone which is passed as an
*     argument.
*
*     Doug DeProspo
*     15Jun95
*     10Jul95 : Modified to allow non-default geoid params.
*
*        Arete Engineering Technologies Corporation
*****************************************************************/
#include <stdio.h>
#include <projects.h>


void contoxy(double lat,double lon,short zone,double *xx,double *yy)



{
   static char *parms[4] ;
   char source1[] = "proj=utm" ;
   char source2[] = "        " ;
   char source3[] = "ellps=WGS84" ;
   char source4[] = "no_defs" ;


   PJ *ref ;
   UV data ;


   parms[0] = source1 ;
   sprintf(source2,"zone=%hd",zone) ;
   parms[1] = source2 ;
   parms[2] = source3 ;
   parms[3] = source4 ;


   if (! (ref = pj_init(4,parms)))
   {
   fprintf(stderr,"Projection Initialization Failed\n") ;
   exit(1) ;
   }

   data.v = lat * DEG_TO_RAD ;
   data.u = lon * DEG_TO_RAD ;
   data = pj_fwd(data,ref) ;
```

```
    *xx = data.u ;
    *yy = data.v ;
    return ;
}
```

```
/*****************************************************************************
*
*    Routine: curfit.c
*
*    This module fits a dipole shape using the MARQUARDT ALGORITHM
*    from Bevington's Data Reduction and Error Analysis(pgs 235-237).
*    The amplitude is scaled out by the norm ratio in the scale subroutine
*
*    Doug DeProspo
*    15Nov94
*    13Feb95 : Protected divide by zero in alpha array. Protected fitting with nfree < 6
*    29Jun95 : Added ampzero as input to arg list
*    10Jul95 : Added earth mag field parameters to argument list
*
*        Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include "dipole.h"


void curfit(int iter,int npts,double *xf,double *yf,double *af,double sigma,double efield[],
        double step[],double ampzero,double *flamda,double param[],double error[],
        double *chisqr,int *eflag)


{

    double temp[nparam],mbeta[nparam],alpha[nparam][nparam];
    double delta,weight,chisq,chisq1,diff ;
    double **tempin,**tempout,**result,**array,**deriv,
        *fplus,*fminus,*b0,*btot ;
    double x0,y0,z0,inc,dec ;
    double dummy ;
    static double m0 ;
    int i,j,k,nfree ;
    int chicheck = 1 ;
    int flag = 0 ;


/*  start main body  */

/*  Allocate dynamic memory for arrays with variable memory and arrays
    passed to invert routine    */

    fplus = dvector(0,npts) ;
    fminus = dvector(0,npts) ;
    b0 = dvector(0,npts) ;
    btot = dvector(0,npts) ;
    array = dmatrix(0,nparam,0,nparam) ;
    deriv = dmatrix(0,nparam,0,npts) ;
    tempout = dmatrix(1,nparam,1,nparam) ;
```

```
/*  Do initializations    */


    nfree = npts-nparam ;
    if (nfree < 6)
      {
      *eflag = 1 ;
      return ;
      }
    weight = 1./(sigma*sigma) ;

    for (i=0; i < nparam ;i++)
      {
      temp[i] = 0. ;
      mbeta[i] = 0. ;
      for (j=0; j < nparam ;j++)
        {
        alpha[i][j] = 0. ;
        }
      }



    getpar(param,&x0,&y0,&z0,&inc,&dec) ;
    btotal2(x0,y0,z0,ampzero,inc,dec,xf,yf,efield,b0,npts) ;
    if (iter == 0){
      dummy = norm(npts,b0);
      m0 = dummy ;  /* initial model amplitude to be used as a rescale parameter */
      }
    else
      {
      scale(m0,npts,b0) ;
      }

/*  Calculate function and chisq derivatives  */

    for (j=0; j < nparam; j++)
      {
      delta = step[j] ;
      param[j] += delta ;
      getpar(param,&x0,&y0,&z0,&inc,&dec) ;
      btotal2(x0,y0,z0,ampzero,inc,dec,xf,yf,efield,fplus,npts) ;
      scale(m0,npts,fplus) ;
      param[j] -= 2.*delta ;
      getpar(param,&x0,&y0,&z0,&inc,&dec) ;
      btotal2(x0,y0,z0,ampzero,inc,dec,xf,yf,efield,fminus,npts) ;
      scale(m0,npts,fminus) ;
      param[j] += delta ;
      for (i=0; i < npts; i++)
        {
        deriv[j][i] = (fplus[i]-fminus[i])/(2.*delta) ;
```

```
        mbeta[j] += weight*((af[i]-b0[i])*deriv[j][i]) ;
          }
       }

     for (j=0; j < nparam; j++)
      {
      for (k=0; k <= j; k++)
          {
       for (i=0; i < npts; i++)
        {
        alpha[j][k] += weight*deriv[j][i]*deriv[k][i] ;
        if (alpha[j][k] == 0.)
          {
          *eflag=1 ;
          return ;
          }
         }
       }
      }
    for (j=0; j < nparam; j++)
     {
     for (k=0; k <= j; k++)
      {
      alpha[k][j] = alpha[j][k] ;
      }
     }

/*  Initial chisq for this iteration */

    chisq1 = chi(af,weight,npts,nfree,b0) ;


    while (chicheck == 1)
     {
     for (j=0; j < nparam; j++)
      {
      for (k=0; k < nparam; k++)
       {
       array[j][k] = alpha[j][k]/sqrt(alpha[j][j]*alpha[k][k]) ;
         }
      array[j][j] = 1. + (*flamda) ;
       }

/*  Invert requires an array which indexes from one  */

    tempin = subdmatrix(array,0,nparam,0,nparam,1,1) ;
    invert(tempin,tempout) ;
    free_subdmatrix(tempin,1,nparam,1,nparam) ;

/*  Convert inverse back to a 0 offset array */
```

```
        result = subdmatrix(tempout,1,nparam,1,nparam,0,0) ;

        for (j=0; j < nparam; j++)
          {
          temp[j] = param[j] ;
          for (k=0; k < nparam; k++)
            {
            temp[j] += (mbeta[k]*result[j][k])/sqrt(alpha[j][j]*alpha[k][k]) ;
              }
          }
        getpar(temp,&x0,&y0,&z0,&inc,&dec) ;
        btotal2(x0,y0,z0,ampzero,inc,dec,xf,yf,efield,btot,npts) ;
        scale(m0,npts,btot) ;
        chisq = chi(af,weight,npts,nfree,btot) ;
        diff = chisq1 - chisq ;

 /*     See if chisq has improved */
        if (diff <  0. )
          {
          *flamda = 10.* (*flamda) ;
          free_subdmatrix(result,0,nparam,0,nparam) ;
          }
        else
          {
          chicheck = 0 ;
          }
      }

      for (j=0; j < nparam; j++)
       {
       error[j] = sqrt(result[j][j]/alpha[j][j] )  ;
       param[j] = temp[j] ;
       }

     *flamda = (*flamda)/10. ;
     *chisqr = chisq ;
     *eflag = flag ;

 /*  Deallocate dynamic memory */

     free_dvector(fplus,0,npts) ;
     free_dvector(fminus,0,npts) ;
     free_dvector(b0,0,npts) ;
     free_dvector(btot,0,npts) ;
     free_dmatrix(array,0,nparam,0,nparam) ;
     free_dmatrix(deriv,0,nparam,0,npts) ;
     free_dmatrix(tempout,1,nparam,1,npts) ;
     free_subdmatrix(result,0,nparam,0,nparam) ;

     return ;
}
```

```
/*****************************************************************************
 *
 *    Routine: dipole.c
 *
 *    This module is the main steering module for the dipole
 *    fitting algorithm. It calls curfit to do the shape fit
 *    and based on the shape results uses ampfit to get the
 *    amplitude.
 *
 *    Doug DeProspo
 *    18Nov 94
 *    13Feb 95 : Added routine checkparam to look for unstable or
 *               unphysical shape fits.
 *    29Jun 95 : Added ampzero as additional input to curvefit routine.
 *    10Jul 95 : Modified routine to pass earth field parameters
 *          Arete Engineering Technologies Corporation
 *****************************************************************************
 */
#include "dipole.h"


void dipole(int npts,double *xf,double *yf,double *af,double sigma,double efield[],
          double initparam[],double param[],double error[],double *amp,double *amperr,
          double *chisq,int *eflag)

{

     double flamda = .001,
          diff,
          chiprev = 99999999.,
          step[nparam] = {.1,.1,.1,.1,.1}, /* step sizes for shape fit params */
          paramprev[nparam] = {0.,0.,0.,0.,0.},
          errorprev[nparam] = {0.,0.,0.,0.,0.},
          ampstep = .01,
          chitest,
          amptest,amperrtest,bmax,amperrz,ampzero ;


     int dofit = 1,
        iter = 0 ,
        i ,
        flag = 0 ,
        checkflag = 0 ;

/*   Main body      */

     for (i=0 ; i < nparam; i++)
      {
      param[i] = initparam[i] ; /* shape parameters set to initial estimates */
      }
```

```
/*    Get an initial amplitude for shapefit  */

      bmax = findmax(af,npts) ;
      bmax = pow(bmax/50000.,.3333333) ;
      ampzero = bmax*param[2] ;

     while (dofit == 1)
        {
/*    This routine fits the dipole shape  */

      curfit(iter,npts,xf,yf,af,sigma,efield,step,ampzero,&flamda,param,
            error,&chitest,&flag) ;
      if (iter >= 100){flag == 1 ;} /* search not converging */
      if (flag == 1 )
       {
       *eflag = flag ;
       return ;
       }
       if (flamda < fcut) flamda = fcut ;


/*    Check chisq with respect to previous iteration */

      diff = 1. - (chitest/chiprev) ;
      if ((diff < tol) && (diff >= 0))
       {
       dofit = 0 ;
       }
      else
       if (diff < 0)
          {

/*    The previous iteration was better  */

        for (i=0; i < nparam; i++)
         {
         param[i] = paramprev[i] ;
         error[i] = errorprev[i] ;
         }
         }
      else
       {
       iter += 1 ;
       chiprev = chitest ;
       for (i=0; i < nparam; i++)
         {
         paramprev[i] = param[i] ;
         errorprev[i] = error[i] ;
         }

         }
```

```
        }
      checkparam(param,initparam,&checkflag) ;
      if (checkflag == 1)
        {
        *eflag = 1 ;
        return ;
        }
      if ((param[4] > 360.) || (param[4] < -360.)) {param[4] = initparam[4];}
      if ((param[3] > 360.) || (param[3] < -360.)) {param[3] = initparam[3];}

/*    Based on shape fit get an initial amplitude estimate   */

      amptest = bmax*param[2] ;

/*    This routine fits dipole amplitude  */

      ampfit(npts,xf,yf,af,param,sigma,efield,ampstep,&amptest,&amperrtest,&chitest) ;
      *chisq = chitest ;
      amperrz = (error[2]/param[2])*amptest ; /* amplitude error from zerror */
      *amperr = pow(amperrtest*amperrtest+ amperrz*amperrz,.5) ; /* add 2 pieces in quadrature */
      *amp = amptest ;
      *eflag = flag ;
      param[2] -= (double)sensorheight ; /* subtract out instrument height */

    return ;
}
```

```
/*****************************************************************************
*
*    INCLUDE FILE: dipole.h
*
*    Main include file for the magproc program
*
*
*    Doug DeProspo
*    16Nov94 : Start with dipole fitting algorithm
*     8Feb95 : Modified file to include prototypes,defines,etc
*              for the automatic threshold detect algorithms and
*              preliminary parameter estimation algorithms
*    10Feb95 : Added utility routine initialization(dvector_init,etc.)
*    15May95 : Add structure typedefs and function prototypes for main
*              input routines (readmag,readnav,readplat)
*    14Jun95 : Added structure typedefs and function prototypes for
*              mapper routines
*    23Jun95 : Started adding prototypes for equalization routines
*    29Jun95 : Started adding output module prototypes
*    11Jul95 : Processor completed
*
*              Arete Engineering Technologies Corporation
*****************************************************************************
*/

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "nrutil.h"

#define nparam 5
#define TINY 1E-20
#define tol .01
#define fcut 1E-8
#define sensorheight 1.5

/* defines for threshold detection algorithm */
#define maxcluster 100
#define maxpoints 4000
#define maxexceed 15000
#define maxdipoles 100
#define cut 7.
#define cut2 17.
#define ncrit 15
#define dexpand 4.

#define maxrec_mag 50000
#define maxrec_nav 50000
#define maxrec_plat 50000

/*    typedefs for structures used in main input routines */
```

```
typedef struct {
    double tzeromag ;
    double dtmag[maxrec_mag] ;
    double ampa[maxrec_mag] ;
    double ampb[maxrec_mag] ;
    double ampc[maxrec_mag] ;
    double ampd[maxrec_mag] ;
  } Umag ;

typedef struct {
    double xyzero[2] ;
    short llzone ;
    double tzeronav ;
    double dtnav[maxrec_nav] ;
    double xtractor[maxrec_nav] ;
    double ytractor[maxrec_nav] ;
    double navstat[maxrec_nav] ;
    double altitude[maxrec_nav] ;
  } Nav ;

typedef struct {
    double tzeroplat ;
    double dtplat[maxrec_plat] ;
    double heading[maxrec_plat] ;
    double vpitch[maxrec_plat] ;
    double vroll[maxrec_plat] ;
    double hpitch[maxrec_plat] ;
    double hroll[maxrec_plat] ;
    double hyaw[maxrec_plat] ;
  } Plat ;

typedef struct {
    double xyzero[2] ;
    short xyzone ;
    double tzeromap ;
    double dtmap[maxrec_mag] ;
    double heading[maxrec_mag] ;
    double xmap[4][maxrec_mag] ;
    double ymap[4][maxrec_mag] ;
    double amp[4][maxrec_mag] ;
  } Map ;


/*    function prototypes for the main input routines */

void readmag(FILE *fpreadmag,Umag *pumag,long *ngoodmag,long *nbadmag) ;

void readnav(FILE *fpreadnav,Nav *punav,long *ngoodnav,long *nbadnav) ;

void readplat(FILE *fpreadplat,Plat *puplat,long *ngoodplat,long *nbadplat) ;
```

```
/*    function prototypes for mapping routines   */

void mapper(Umag *pumag,Nav *pnav,Plat *pplat,long ngoodmag,long ngoodnav,long ngoodplat,Map *pmap) ;

void contoxy(double lat,double lon,short zone,double *x,double *y) ;

void getzone(double lon,short *zone) ;

void getxytractor(double navtime,Nav *pnav,long ngoodnav,long inavlow,
            long *navindex1,long *navindex2) ;

void getdxymags(double plattime,Plat *pplat,long ngoodplat,long iplatlow,
            long *platindex,double dx[],double dy[]) ;


/*    function prototypes for equalization and instrument check routines   */

void headingindex(Map *pmap,long ngoodmap,int *dirflag,long *nhead1,long *nhead2) ;

void getstats(Map *pmap,long ngoodmap,int *dirflag,long nhead1,long nhead2,
          double insmedian[],double insrms[]) ;

int intcmp(const void *v1,const void *v2) ;

void inscheck(double insrms[],int insflag[]) ;

void combinedata(Map *pmap,long ngoodmap,int insflag[],int *dirflag,
            double insmedian[],double insrms[],double *xdata,double *ydata,
            double *ampdata,long *ncount,double *sigma) ;


/*    function prototypes for output routines   */

void contoll(double x,double y,short zone,double *lat,double *lon) ;

void outstd(FILE * fpstd,short zone,int fitflag,int i,double parm[],double error[],
        double amp,double amperr,double chisq,double xyref[]) ;

void conmin(double l,double *ldeg,double *lmin) ;

void getconf(double size,double sizeerr,double bound[2],double *conf) ;


/*    function prototypes for main dipole routines   */

void dipole(int npts,double *xf,double *yf,double *af,double sigma,double field[],double param[],
        double initparam[],double error[],double *amp,double *amperr,double *chisq,
        int *eflag) ;
```

```
void curfit(int iter,int npts,double *xf,double *yf,double *af,double sigma,double field[],double step[]
        ,double ampzero,double *flamda,double param[],double error[],double *chisqr,int *eflag) ;

void ampfit(int npts,double *xf,double *yf,double *af,double param[],double sigma,double field[],double ampstep,
        double *amp,double *amperr,double *chisq) ;



/*    function prototypes for threshold detection and prelim parameter est routines   */

void exceed(double *xdata,double *ydata,double *ampdata,long ntot,int csign,double thresh,
        double *xexceed,double *yexceed,double *aexceed,int *nexceed) ;

void cluster(int nexceed,double *xexceed,double *yexceed,double *aexceed,int *ncluster,
        int *count,double **xcluster,double **ycluster,double **acluster) ;

void centroid(int ncluster,int *count,double **xcluster,double **ycluster,double **acluster,
        double *cenx,double *ceny) ;

void assoc(double *cenxp,double *cenyp,double *cenxm,double *cenym,int nclusterp,
        int nclusterm,int *ndipole,int *indexp,int *indexm) ;

void initparams(double **xclusterm,double **yclusterm,double **xclusterp,double **yclusterp,
        double **aclusterm,double **aclusterp,double *cenxp,double *cenyp,
        double *cenxm,double *cenym,int *indexm,int *indexp,int ndip,int *countp,
        int *countm,double initparam[]) ;

double estinc(double amax,double amin) ;

double estz(double **x,double **y,double **amp,int *count,int *index,int ndip) ;

void prepdipole(double **xclusterm,double **yclusterm,double **xclusterp,double **yclusterp,
        double **aclusterm,double **aclusterp,int *indexm,int *indexp,int ndip,
        int *countp,int *countm,double *xf,double *yf,double *af,int *npts) ;

void trim(double *xin,double *yin,double *ampin,int nout,double *xout,double *yout,
        double *ampout) ;

void expand(double initparam[],double *xdata,double *ydata,double *ampdata,long ntot,
        double *xcon,double *ycon,double *ampcon,int *nptcon) ;

void checkparam(double param[],double initparam[],int *eflag) ;



/*    function prototypes for fitutil.c routines   */

double chi(double *af,double weight,int npts,int nfree,double *b0) ;
void scale(double m0,int npts,double *b0) ;
double norm(int npts,double *b0) ;
```

```
double findmax(double *af,int npts) ;
double findmin(double *af,int npts) ;
void getpar(double param[],double *x0,double *y0,double *z0,double *inc,double *dec) ;
void invert(double **a,double **y) ;
void ludcmp(double **a,int n, int *indx, double *d) ;
void lubksb(double **a,int n, int *indx, double b[]) ;
void btotal2(double x0,double y0,double z0,double a0,double inc,double dec,
        double *x,double *y,double field[],double *btot,int npts) ;
void dvector_init(double *a,int size,double value) ;
void ldvector_init(double *a,long size,double value) ;
void ivector_init(int *a,int size,int value) ;
void livector_init(int *a,long size,int value) ;
void dmatrix_init(double **a,int size1,int size2,double value) ;
void imatrix_init(int **a,int size1,int size2,int value) ;


/*   function prototypes for numerical recipe like routines that did not
     exist in double precision(in fitutil.c)  */

double **subdmatrix(double **a,int oldrl,int oldrh,int oldcl,int oldch,int newrl,int newcl) ;
void free_subdmatrix(double **b,int nrl,int nrh,int ncl,int nch) ;

int *livector(int nl, long nh);
double *ldvector(int nl, long nh );
void free_livector(int *v, int nl, long nh );
void free_ldvector( double *v, int nl, long nh);
```

```
/*****************************************************************************
*
*    Routine: estinc.c
*
*    This module estimates an inclination angle based on
*    the amplitude ratio of the positive and negative peaks
*
*    Doug DeProspo
*    27Jan95
*        Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include "dipole.h"

double estinc(double amax,double amin)


{

    double inc,dummy ;
    int ratio ;

/*  start main body  */



    dummy = (amax/amin)*100. ;
    ratio = (int)dummy ;
    if ((ratio <= 0) && (ratio > -5)) {ratio = 0 ;}
    if ((ratio <= -5) && (ratio > -20))  {ratio = 1 ;}
    if ((ratio <= -20) && (ratio > -60))  {ratio = 2 ;}
    if ((ratio <= -60) && (ratio > -120)) {ratio = 3 ;}
    if ((ratio <= -120) && (ratio > -400)) {ratio = 4 ;}
    if (ratio <= -400)  {ratio = 5 ;}
    switch (ratio)
      {
      case 0:
        {
        inc = -80. ;
        break ;
        }
      case 1:
        {
        inc = -50. ;
        break ;
        }
      case 2:
        {
        inc = -20. ;
        break ;
        }
```

```
        case 3:
          {
          inc = 10. ;
          break ;
          }
        case 4:
          {
          inc = 40. ;
          break ;
          }
        case 5:
          {
          inc = 80. ;
          break ;
          }
        }

    return inc ;
}
```

```
/*****************************************************************************
*
*    Routine: estz.c
*
*    This module estimates a depth for a single + or -
*    lobe when there is no opposite sign lobe.
*    Uses a normalized radial gradient to estimate depth.
*
*    Doug DeProspo
*    31Jan95
*        Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include "dipole.h"

double estz(double **x,double **y,double **amp,int *count,
        int *index,int ndip)



{

    double zest,maxamp,minamp ;
    double dx2,dy2,dist,grad ;
    int points,i,j,num,imax,imin ;
    int igrad,dummy ;

/*  start main body  */


    num = index[ndip] ;
    points = count[num] ;
    if (points == 1)
      {
/*  only a single point- set depth shallow and return */
      zest = 1. ;
      return zest ;
      }

/*  find data maximum */
    maxamp = 0. ;
    for (i=0;i < points ;i++)
      {
      if (fabs(amp[num][i]) >= maxamp)
        {
        maxamp = fabs(amp[num][i]) ;
        imax = i ;
        }
      }

/*  find data minimum */
```

```
minamp = 99999999. ;
for (i=0;i < points ;i++)
  {
  if (fabs(amp[num][i]) <= minamp)
    {
    minamp = fabs(amp[num][i]) ;
    imin = i ;
    }
  }

/*   normalize max and min */

minamp = minamp/maxamp ;
maxamp = 1. ;
dx2 = pow(x[num][imax]-x[num][imin],2.) ;
dy2 = pow(y[num][imax]-y[num][imin],2.) ;
dist = sqrt(dx2 + dy2) ;
grad = (maxamp-minamp)/dist ; /* have normalized gradient between max and min */

/*  go through lookup table to assign depth */
dummy =(int)(grad*100.) ;

if (dummy > 33) {igrad = 0;}
if ((dummy > 16) && (dummy <= 33)) {igrad = 1;}
if ((dummy > 8) && (dummy <= 16)) {igrad = 2;}
if ((dummy > 4) && (dummy <= 8)) {igrad = 3;}
if (dummy <= 4)  {igrad = 4;}

switch (igrad)
  {
  case 0:
    {
    zest = 3. ;
    break ;
    }
  case 1:
    {
    zest = 5. ;
    break ;
    }
  case 2:
    {
    zest = 10. ;
    break ;
    }
  case 3:
    {
    zest = 15. ;
    break ;
    }
  case 4:
```

```
        {
        zest = 20. ;
        break ;
        }
      }

    return zest ;
}
```

```
/*****************************************************************************
*
*    Routine: exceed.c
*
*    This module takes input x,y,amp vectors, thresholds the amplitude,
*    and creates output x,y,amp vectors of points over threshold.
*
*    Doug DeProspo
*    21Dec94
*        Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include "dipole.h"


void exceed(double *xdata,double *ydata,double *ampdata,long ntot,int csign,
          double thresh,double *xexceed,double *yexceed,double *aexceed,
           int *nexceed)



{

    int ngood=0 ;
    long i ;


    if (csign == 1)
      {
      for (i=0; i < ntot;i++)
          {
        if (ampdata[i] > thresh)
          {
          xexceed[ngood]=xdata[i] ;
          yexceed[ngood]=ydata[i] ;
          aexceed[ngood]=ampdata[i] ;
          ngood += 1 ;
          }
        }
      }
    else
      {
      for (i=0; i < ntot;i++)
          {
        if (ampdata[i] < thresh)
          {
          xexceed[ngood]=xdata[i] ;
          yexceed[ngood]=ydata[i] ;
          aexceed[ngood]=ampdata[i] ;
          ngood += 1 ;
          }
```

```
    }
  }

  *nexceed = ngood ;
  return ;

}
```

```
/*****************************************************************************
*
*    Routine: expand.c
*
*    This module expands the number of points associated to a
*    dipole(if npts < ncrit) by including all points in a region
*    defined by a circle of radius dmax +dexpand, where dmax is the point
*    that is the furthest from the dipole centroid.
*    Doug DeProspo
*    31Jan95
*        Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include "dipole.h"

void expand(double initparam[],double *xdata,double *ydata,double *ampdata,long ntot,
        double *xcon,double *ycon,double *ampcon,int *nptcon)



{


    double dmax =0.,xcen,
          ycen,dx2,dy2,
          dist ;
    int points,newpoints=0,imax ;
    long i ;

/*  start main body  */

    points = *nptcon ;

    if (points > 1)
      {
      xcen = initparam[0] ;
      ycen = initparam[1] ;

      for (i=0;i < points; i++)
        {
        dx2 = pow(xcon[i]-xcen,2.) ;
        dy2 = pow(ycon[i]-ycen,2.) ;
        dist =sqrt(dx2+dy2) ;
        if (dist >= dmax)
          {
          dmax = dist ;
          imax = i ;
          }
        }
      }
    else
```

```
        {
        xcen = xcon[0] ;
        ycen = ycon[0] ;
        }

      for (i=0;i < ntot ;i++)
        {
        dx2 = pow(xdata[i]-xcen,2.) ;
        dy2 = pow(ydata[i]-ycen,2.) ;
        dist = sqrt(dx2+dy2) ;
        if ( dist < dmax + dexpand)
          {
          xcon[newpoints] = xdata[i] ;
          ycon[newpoints] = ydata[i] ;
          ampcon[newpoints] = ampdata[i] ;
          newpoints += 1 ;
          }
        }

      *nptcon = newpoints ;
      return  ;
}
```

```
/*****************************************************************************
*
*     Routine: fitutil.c
*
*     This file contains the utility routines chi,scale,norm ,getpar,findmax,finmin,invert,
*     ludcmp,lubksb(the last 3 routines are from Numerical Recipes in C(pgs44-45).
*
*     Doug DeProspo
*     16Nov94
*     10Feb95 : Added utility routines ivector_init,dvector_init,imatrix_init,dmatrix_init
*          Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include "dipole.h"


double chi(double *af,double weight,int npts,int nfree,double *b0)
{
    int i ;
    double chisq = 0. ;

    for (i=0; i < npts ;i++)
      {
      chisq += pow(af[i]-b0[i],2) ;
      }
    chisq = chisq*(weight/nfree) ;

    return chisq ;
}

void scale(double m0,int npts,double *b0)
{
    double factor,m1=0. ;
    int i ;

    m1 = norm(npts,b0) ;
    if (m1 <= 0.)
      {
      factor = 1. ;
      }
    else
      {
      factor = m0/m1 ;
      }
    for (i=0; i < npts ;i++)
      {
      b0[i] = b0[i]*factor ;
      }

   return ;
}
```

```
double norm(int npts,double *b0)
{
    double m1 = 0. ;
    int i ;

    for (i=0; i < npts ;i++)
      {
      m1 += b0[i]*b0[i] ;
      }
    m1 = sqrt(m1) ;

    return m1 ;
}

double findmax(double *af,int npts)
{
    double bmax = -99999. ;
    int i ;

    for (i=0; i < npts ;i++)
      {
      if (fabs(af[i]) > bmax) bmax = fabs(af[i]) ;
      }

     return bmax ;
}
double findmin(double *af,int npts)
{
    double bmin = 99999. ;
    int i ;

    for (i=0; i < npts ;i++)
      {
      if (af[i] < bmin) bmin = af[i] ;
      }

     return bmin ;
}

void getpar(double param[],double *x0,double *y0,double *z0,double *inc,double *dec)
{
    *x0 = param[0] ;
    *y0 = param[1] ;
    *z0 = param[2] ;
    *inc = param[3] ;
    *dec = param[4] ;

   return ;
}

void invert(double **a,double **y)
```

```
{
    int i,j,*indx,n ;
    double d,*col ;

    n = nparam ;
    col = dvector(1,n);
    indx = ivector(1,n);
    ludcmp(a,n,indx,&d) ;
    for (j=1;j <= n ; j++)
      {
      for (i=1; i <= n; i++) col[i] = 0. ;
      col[j] = 1. ;
      lubksb(a,n,indx,col) ;
      for (i=1; i <=n; i++) y[i][j] = col[i] ;
      }

   free_dvector(col,1,n) ;
   free_ivector(indx,1,n) ;
   return ;
}

void ludcmp(double **a, int n, int *indx, double *d)
{
        int i,imax,j,k;
        double big,dum,sum,temp;
        double  *vv;

        vv=dvector(1,n);
        *d=1.0;

        for (i=1;i<=n;i++) {
                big=0.0;
                for (j=1;j<=n;j++)
                        if ((temp=fabs(a[i][j])) > big) big=temp;
                if (big == 0.0) nrerror("Singular matrix in routine ludcmp");
                vv[i]=1.0/big;
        }


        for (j=1;j<=n;j++) {
                for (i=1;i<j;i++) {
                        sum=a[i][j];
                        for (k=1;k<i;k++) sum -= a[i][k]*a[k][j];
                        a[i][j]=sum;

                }
                big=0.0;
                for (i=j;i<=n;i++) {
                        sum=a[i][j];
                        for (k=1;k<j;k++)
                                sum -= a[i][k]*a[k][j];
```

```
                        a[i][j]=sum;
                        if ( (dum=vv[i]*fabs(sum)) >= big) {
                                big=dum;
                                imax=i;
                        }
                }
                if (j != imax) {
                        for (k=1;k<=n;k++) {
                                dum=a[imax][k];
                                a[imax][k]=a[j][k];
                                a[j][k]=dum;
                        }
                        *d = -(*d);
                        vv[imax]=vv[j];
                }
                indx[j]=imax;
                if (a[j][j] == 0.0) a[j][j]=TINY;
                if (j != n) {
                        dum=1.0/(a[j][j]);
                        for (i=j+1;i<=n;i++) a[i][j] *= dum;
                }
        }
        free_dvector(vv,1,n);
}
/* (C) Copr. 1986-92 Numerical Recipes Software !N#. */

void lubksb(double **a, int n, int *indx, double b[])
{
        int i,ii=0,ip,j;
        double sum;

        for (i=1;i<=n;i++) {
                ip=indx[i];
                sum=b[ip];
                b[ip]=b[i];
                if (ii)
                        for (j=ii;j<=i-1;j++) sum -= a[i][j]*b[j];
                else if (sum) ii=i;
                b[i]=sum;
        }
        for (i=n;i>=1;i--) {
                sum=b[i];
                for (j=i+1;j<=n;j++) sum -= a[i][j]*b[j];
                b[i]=sum/a[i][i];
        }
}
/* (C) Copr. 1986-92 Numerical Recipes Software !N#. */


double **subdmatrix( double **a, int oldrl, int oldrh, int oldcl, int oldch,
```

```
  int newrl, int newcl)
 {
        int i,j;
        double **m;

        m=(double **) malloc((unsigned) (oldrh-oldrl+1)*sizeof(double*));
        if (!m) nrerror("allocation failure in submatrix()");
        m -= newrl;

        for(i=oldrl,j=newrl;i<=oldrh;i++,j++) m[j]=a[i]+oldcl-newcl;

        return m;
}


void free_subdmatrix( double **b, int nrl, int nrh, int ncl, int nch)
{
        free((char*) (b+nrl));
}

void ivector_init(int *a,int size,int value)
{
      int i ;

      for (i=0;i < size;i++)
        {
        a[i] = value ;
        }

      return ;
}
void livector_init(int *a,long size,int value)
{
      long i ;

      for (i=0;i < size;i++)
        {
        a[i] = value ;
        }

      return ;
}
void dvector_init(double *a,int size,double value)
{
      int i ;

      for (i=0;i < size;i++)
        {
        a[i] = value ;
        }
```

```
        return ;
}
void ldvector_init(double *a,long size,double value)
{
        long i ;

        for (i=0;i < size;i++)
          {
          a[i] = value ;
          }

        return ;
}

void imatrix_init(int **a,int size1,int size2,int value)
{

        int i,j ;

        for (i=0; i < size1;i++)
          {
          for (j=0; j < size2;j++)
            {
            a[i][j] = value ;
            }
          }

        return ;
}

void dmatrix_init(double **a,int size1,int size2,double value)
{

        int i,j ;

        for (i=0; i < size1;i++)
          {
          for (j=0; j < size2;j++)
            {
            a[i][j] = value ;
            }
          }

        return ;
}
```

```
/*****************************************************************************
*
*     Routine: getconf.c
*
*     This routine calculates a confidence level for a given size
*     classification from the ratio of distance to nearest size boundary
*     to the stdev of the size estimate.
*
*
*     Doug DeProspo
*     7July95
*          Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include "dipole.h"


void getconf(double size,double sizeerr,double bound[],double *rconf)


{

     double delta,delta1,delta2,ratio,conf ;


     if (size <= bound[0])
       {
       delta = bound[0] - size ;
       }
     else
       {
       if (size >= bound[1])
         {
         delta = size - bound[1] ;
         }
       else
         {
         delta1 = bound[1] - size ;
         delta2 = size - bound[0] ;
         if ( delta1 <= delta2)
           {
           delta = delta1 ;
           }
         else
           {
           delta = delta2 ;
           }
         }
       }
```

```
    ratio = delta / sizeerr ; /* number of stdev units from closest boundary */

/*  Now calculate confidence from a lookup table */

    if (ratio <= .06){conf = .01 ;}
    if ((ratio > .06) && (ratio <= .19)){conf = .05 ;}
    if ((ratio > .19) && (ratio <= .31)){conf = .15 ;}
    if ((ratio > .31) && (ratio <= .45)){conf = .25 ;}
    if ((ratio > .45) && (ratio <= .6)){conf = .35 ;}
    if ((ratio > .6) && (ratio <= .75)){conf = .45 ;}
    if ((ratio > .75) && (ratio <= .94)){conf = .55 ;}
    if ((ratio > .94) && (ratio <= 1.15)){conf = .65 ;}
    if ((ratio > 1.15) && (ratio <= 1.45)){conf = .75 ;}
    if ((ratio > 1.45) && (ratio <= 2.0)){conf = .85 ;}
    if ((ratio > 2.0) && (ratio <= 2.5)){conf = .95 ;}
    if (ratio >= 2.5){conf = .99 ;}

    conf *= 100. ; /* convert to percent */

    *rconf = conf ;

    return ;
}
```

```
/*****************************************************************************
*
*    Routine: getdxymags.c
*
*    This module returns dx and dy which contain the
*    offsets in x and y from the gps antenna on back
*    of tractor for each mag
*
*    Doug DeProspo
*    14Jun95
*
*        Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include "dipole.h"

#define dtr (3.14159/180.)


void getdxymags(double plattime,Plat *pplat,long ngoodplat,long iplatlow,
            long *platindex,double dx[],double dy[])


{

     double dela[] = {-304.184,29.625,2.238,1.} ;
     double delb[] = {-303.689,9.625,1.595,1.} ;
     double delc[] = {-303.689,-9.625,1.533,1.} ;
     double deld[] = {-304.426,-29.625,2.763,1.} ;
     double d1[4],d2[4] ;
     double pa,ya,ra,hp,hr,hy,head,coshd,sinhd ;
     double cospa,cosya,cosra,sinpa,sinya,sinra ;
     double t1,t2,t3,t4 ;
     double time,timeprev,dt1,dt2 ;
     long i,iwant,isave1,isave2 ;
     short flag = 0 ;

    i = iplatlow ;
    timeprev = pplat->dtplat[i] ;

    while (flag == 0)
        {
      if (i < ngoodplat)
        {
        time = pplat->dtplat[i] ;
        if ((time > plattime) && (plattime > timeprev))
          {
          flag = 1 ;
          isave1 = i-1 ;
          isave2 = i ;
          }
```

```
            else
              {
              i += 1 ;
              timeprev = time ;
              }
            }
          else
            {
            flag = 1 ;
            }
          }

        dt1 = fabs((pplat->dtplat[isave1]) - plattime) ;
        dt2 = fabs((pplat->dtplat[isave2]) - plattime) ;

      if (dt1 <= dt2)
        {
        iwant = isave1 ;
        }
      else
        {
        iwant = isave2 ;
        }
      *platindex = isave1 ;

  /*  Extract hpitch,hroll,hyaw and tractor heading */

        hp = pplat->hpitch[iwant] ;
        hr = pplat->hroll[iwant] ;
        hy = pplat->hyaw[iwant] ;
        head = pplat->heading[iwant] ;

        pa = (360. - hp) * dtr ;
        ya = hy * dtr ;
        ra = hr * dtr ;
        cospa = cos(pa) ;
        cosya = cos(ya) ;
        cosra = cos(ra) ;
        sinpa = sin(pa) ;
        sinya = sin(ya) ;
        sinra = sin(ra) ;

  /* 1st row */

        t1 = cospa * cosya ;
        t2 = -cospa * sinya * cosra + sinpa * sinra ;
        t3 = cospa * sinya * sinra + sinpa * cosra ;
        t4 = -14.188 * cospa * cosya - 5. * cospa - 49.55 ;

        d1[0] = t1 * dela[0] + t2 * dela[1] + t3 * dela[2] + t4 * dela[3] ;
        d1[1] = t1 * delb[0] + t2 * delb[1] + t3 * delb[2] + t4 * delb[3] ;
```

```
        d1[2] = t1 * delc[0] + t2 * delc[1] + t3 * delc[2] + t4 * delc[3] ;
        d1[3] = t1 * deld[0] + t2 * deld[1] + t3 * deld[2] + t4 * deld[3] ;

    /* 2nd row */

        t1 = sinya ;
        t2 = cosya * cosra ;
        t3 = -cosya * sinra ;
        t4 = -14.188 * sinya - 0.344 ;

        d2[0] = t1 * dela[0] + t2 * dela[1] + t3 * dela[2] + t4 * dela[3] ;
        d2[1] = t1 * delb[0] + t2 * delb[1] + t3 * delb[2] + t4 * delb[3] ;
        d2[2] = t1 * delc[0] + t2 * delc[1] + t3 * delc[2] + t4 * delc[3] ;
        d2[3] = t1 * deld[0] + t2 * deld[1] + t3 * deld[2] + t4 * deld[3] ;

    /* Heading Correction and Conversion from inches to ft. */

        coshd = cos(head * dtr) ;
        sinhd = sin(head * dtr) ;

        dx[0] = (d1[0] * coshd - d2[0] * sinhd) /12. ;
        dy[0] = (d2[0] * coshd + d1[0] * sinhd) /12. ;
        dx[1] = (d1[1] * coshd - d2[1] * sinhd) /12. ;
        dy[1] = (d2[1] * coshd + d1[1] * sinhd) /12. ;
        dx[2] = (d1[2] * coshd - d2[2] * sinhd) /12. ;
        dy[2] = (d2[2] * coshd + d1[2] * sinhd) /12. ;
        dx[3] = (d1[3] * coshd - d2[3] * sinhd) /12. ;
        dy[3] = (d2[3] * coshd + d1[3] * sinhd) /12. ;

        return ;

}
```

```
/*****************************************************************************
*
*     Routine: getstats.c
*
*     This module finds the median and rms levels for the
*     four SOCS mags for the two dominant headings. The
*     rms is determined from the 31st and 69th percentile
*     points.
*
*
*     Doug DeProspo
*     26Jun95
*          Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include "dipole.h"


void getstats(Map *pmap,long ngoodmap,int *dirflag,long nhead1,long nhead2,
              double insmedian[],double insrms[])



{

     long ndir1,ndir2,lowindex1,lowindex2,highindex1,highindex2 ;
     long medindex1,medindex2,i,k ;
     double *amp1,*amp2 ;

     amp1 = ldvector(0,nhead1) ;
     amp2 = ldvector(0,nhead2) ;
     ldvector_init(amp1,nhead1,0.) ;
     ldvector_init(amp2,nhead2,0.) ;

     for (i=0 ; i < 4 ; i++)
          {
       ndir1 = 0 ;
       ndir2 = 0 ;
       for (k=0 ; k < ngoodmap ; k++)
            {
         if (dirflag[k] == 1)
           {
           amp1[ndir1] = pmap->amp[i][k] ;
           ndir1 += 1 ;
           }
         else
           {
           amp2[ndir2] = pmap->amp[i][k] ;
           ndir2 += 1 ;
           }
```

```
        }
      medindex1 = ndir1/2 ;
      medindex2 = ndir2/2 ;
      lowindex1 = medindex1 - (long)(.19 * ndir1) ;
      lowindex2 = medindex2 - (long)(.19 * ndir2) ;
      highindex1 = medindex1 + (long)(.19 * ndir1) ;
      highindex2 = medindex2 + (long)(.19 * ndir2) ;
      qsort(amp1,ndir1,sizeof(amp1[0]),intcmp) ;
      qsort(amp2,ndir2,sizeof(amp2[0]),intcmp) ;
      insmedian[i] = amp1[medindex1] ;
      insrms[i] = amp1[highindex1] - amp1[lowindex1] ;
      insmedian[i+4] = amp2[medindex2] ;
      insrms[i+4] = amp2[highindex2] - amp2[lowindex2] ;
      }

    free_ldvector(amp1,0,ndir1) ;
    free_ldvector(amp2,0,ndir2) ;

    return ;


}
```

```
/*****************************************************************************
*
*     Routine: getxytractor.c
*
*     This module returns the two nav records closest in
*     time to the mag time.
*
*     Doug DeProspo
*     14Jun95
*
*          Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include "dipole.h"


void getxytractor(double navtime,Nav *pnav,long ngoodnav,long inavlow,
                  long *navindex1,long *navindex2)



{

     double time,timeprev ;
     long istart,i ;


     istart = inavlow ;
     timeprev = pnav->dtnav[istart] ;

     for (i=istart ; i < ngoodnav ; i++)
         {
      time = pnav->dtnav[i] ;
      if ((time > navtime) && (timeprev < navtime))
        {
        *navindex1 = i-1 ;
        *navindex2 = i ;
        return ;
        }
      timeprev = time ;
      }

    *navindex1 = -1 ; /*  error condition */
    *navindex2 = -1 ; /*  error condition */

    return ;

}
```

```
/*******************************************************************************
*
*     Routine: getzone.c
*
*     This routine calculates the UTM zone for uses in the lat/long
*     conversion program.
*
*
*     Doug DeProspo
*     10July95
*          Arete Engineering Technologies Corporation
*******************************************************************************
*/
#include "dipole.h"


void getzone(double lon,short *zone)


{

     short mzone ;

     if ((lon > 66.) && (lon <=72.)){mzone = 19 ;}
     if ((lon > 72.) && (lon <=78.)){mzone = 18 ;}
     if ((lon > 78.) && (lon <=84.)){mzone = 17 ;}
     if ((lon > 84.) && (lon <=90.)){mzone = 16 ;}
     if ((lon > 90.) && (lon <=96.)){mzone = 15 ;}
     if ((lon > 96.) && (lon <=102.)){mzone = 14 ;}
     if ((lon > 102.) && (lon <=108.)){mzone = 13 ;}
     if ((lon > 108.) && (lon <=114.)){mzone = 12 ;}
     if ((lon > 114.) && (lon <=120.)){mzone = 11 ;}
     if ((lon > 120.) && (lon <=126.)){mzone = 10 ;}

     *zone = mzone ;

    return ;
}
```

```
/*****************************************************************************
*
*     Routine: initparams.c
*
*     This module takes a dipole and estimates,where possible,the
*     dipole shape parameters(x,y,z,inc,dec) to be used
*     as initial values for the dipole fitting routine.
*
*     Doug DeProspo
*     22Dec94
*     29Jun95 : Turned off estz routine for single-pole fits-hardwire
*               shallow estimate (4 ft.)
*          Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include "dipole.h"


void initparams(double **xclusterm,double **yclusterm,double **xclusterp,double **yclusterp,
                double **aclusterm,double **aclusterp,double *cenxp,double *cenyp
                ,double *cenxm,double *cenym,int *indexm,int *indexp,int ndip,
                int *countp,int *countm,double initparam[])



{

       double dx,dy,dx2,dy2 ;
       double amax,amin ;
       double *apos,*aneg ;
       int numpos,numneg ;
       int i ;


     if ((indexp[ndip] != -1) && (indexm[ndip] != -1))
/*   2 pole dipole */
       {
       numpos = indexp[ndip] ;
       numneg = indexm[ndip] ;

       apos = dvector(0,countp[numpos]) ;
       dvector_init(apos,countp[numpos],0.) ;

       aneg = dvector(0,countm[numneg]) ;
       dvector_init(aneg,countm[numneg],0.) ;

       for (i=0; i <countp[numpos] ;i++)
           {
        apos[i] = aclusterp[numpos][i] ;
        }
       for (i=0; i <countm[numneg] ;i++)
```

```
            {
           aneg[i] = aclusterm[numneg][i] ;
           }
         amax = findmax(apos,countp[numpos]) ;
         amin = findmin(aneg,countm[numneg]) ;

         free_dvector(apos,0,countp[numpos]) ;
         free_dvector(aneg,0,countm[numneg]) ;

         initparam[0] = (cenxp[indexp[ndip]] + cenxm[indexm[ndip]])/2. ; /* xest */
         initparam[1] = (cenyp[indexp[ndip]] + cenym[indexm[ndip]])/2. ; /* yest */
         dx = cenxm[indexm[ndip]] -cenxp[indexp[ndip]] ;
         dy = cenym[indexm[ndip]] -cenyp[indexp[ndip]] ;
         dx2 = dx*dx ;
         dy2 = dy*dy ;
         initparam[2] = .75*pow(dx2+dy2,.5) ; /* zest */
         initparam[3] = estinc(amax,amin) ; /* inclination est */
         initparam[4] =atan2(dy,dx) * (180./3.14159) ; /* declination estimate */
         }
       else
         {
         if (indexm[ndip] == -1)
/*     single + pole  */
           {
           initparam[0] = cenxp[indexp[ndip]] ;
           initparam[1] = cenyp[indexp[ndip]] ;
/*           initparam[2] = estz(xclusterp,yclusterp,aclusterp,countp,indexp,ndip) ; */
           initparam[2] = 4. ;
           initparam[3] = 80. ;
           initparam[4] = 0. ;
           }
         else
/*     single - pole  */
           {
           initparam[0] = cenxm[indexm[ndip]] ;
           initparam[1] = cenym[indexm[ndip]] ;
/*           initparam[2] = estz(xclusterm,yclusterm,aclusterm,countm,indexm,ndip) ; */
           initparam[2] = 4. ;
           initparam[3] = -80. ;
           initparam[4] = 0. ;
           }
         }

       return ;
}
```

```
/*****************************************************************************
*
*    Routine: inscheck.c
*
*    This routine uses a comparison of the sensor rms to
*    determine if there is a bad sensor. It cannot detect
*    the presence of more than one.
*
*    Doug DeProspo
*    27Jun95
*        Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include "dipole.h"
#define factor 2.


void inscheck(double insrms[],int insflag[])


{
     int imax,imin,i ;
     double amax,amin,rmsmean,rmsdev1,rmsdev2,rmssum ;
     double temp[4] ;


     for (i=0 ; i < 4 ; i++)
         {
       temp[i] = insrms[i] ;
       }

     amax  = findmax(temp,4) ;
     amin = findmin(temp,4) ;

     for (i=0 ; i<4 ; i++)
         {
       if (amax == temp[i]){imax=i ;}
       if (amin == temp[i]){imin=i ;}
       }

     if ((amax/amin) < factor) /* rms are within a factor of 2 */
       {
       return ;
       }
     else /* problem with one or more instrument- assume one bad */
       {
/* calculate dispersion amongst rms without imin */

       rmssum = 0. ;
       for (i=0 ; i<4 ; i++)
         {
```

```
        if (i != imin)
          {
          rmssum += insrms[i] ;
            }
        }
      rmsmean = rmssum/3. ;
      rmsdev1 = 0. ;
      for (i=0 ; i<4 ; i++)
        {
        if (i != imin)
          {
          rmsdev1 += pow((rmsmean - insrms[i]),2.) ;
            }
        }

      rmsdev1 = sqrt(rmsdev1/2.) ;

/* calculate dispersion amongst rms without imax */

      rmssum = 0. ;
      for (i=0 ; i<4 ; i++)
        {
        if (i != imax)
          {
          rmssum += insrms[i] ;
            }
        }
      rmsmean = rmssum/3. ;
      rmsdev2 = 0. ;
      for (i=0 ; i<4 ; i++)
        {
        if (i != imax)
          {
          rmsdev2 += pow((rmsmean - insrms[i]),2.) ;
            }
        }
      rmsdev2 = sqrt(rmsdev2/2.)  ;

      if (rmsdev2 >= rmsdev1)
        {
        insflag[imin] = 1 ;
        insflag[imin+4] = 1 ;
        }
      else
        {
        insflag[imax] = 1 ;
        insflag[imax+4] = 1 ;
        }
      }
    return ;
}
```

```
/*****************************************************************************
*
*     Routine: intcmp.c
*
*
*     Doug DeProspo
*     26Jun95
*          Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include "dipole.h"


int intcmp(const void *v1,const void *v2)


{
    return (*(int *)v1 - *(int *)v2) ;
}
```

```
/*****************************************************************************%
*
*     Routine: magproc.c
*
*     This module is the main steering module for the automatic
*     SOCS magnetometer processing chain.
*
*
*     Doug DeProspo
*     21Nov 94: Start with dipole fitting algorithm
*     21Dec 94: Added threshold based cluster detector and coupled this
*               to dipole fitting algorithm
*     26April 95: Added input routines to read in mag,nav and platform
*                 routines.
*     14June 95: Added mapper to do mapping to ground based coordinates.
*     25June 95: Added routines to perform median,rms estimation and
*                 correction for directional bias(equalization).
*     30June 95: Added output routines
*      7July 95: changed some routines to handle long variables
*     10July 95: Now reads param file for earth mag field params
*     11July 95: Version 1.0 of processor completed
*
*          Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include "dipole.h"


main()

{

      double param[nparam] ={0.,0.,0.,0.,0.},
           error[nparam] = {0.,0.,0.,0.,0.},
           initparam[nparam] = {0.,0.,0.,0.,0.} ;
      double chisq,amp,amperr,
           sigma,
           threshplus,threshminus ;
      double *xdata,*ydata,*ampdata,
            *xexceedp,*yexceedp,*aexceedp,
            *xexceedm,*yexceedm,*aexceedm,
            *xcon,*ycon,*ampcon,
            *xf,*yf,*af,
            *cenxp,*cenyp,*cenxm,*cenym,
            **xclusterp,**yclusterp,
            **xclusterm,**yclusterm,
            **aclusterp,**aclusterm ;

      double insmedian[8],insrms[8],xyref[2] ;

      double efield[2] = {65.,0.} ; /* DEFAULTS FOR EARTH FIELD */
```

```
double *debx,*deby,*fitx,*fity,*fitz,*fiti,*fitd,*fitamp,*fitchi ;

int dummy1,dummy2 ;

int nexceedp,nexceedm,
    nclusterm,nclusterp,ndipole,
    csign,nsize,nptcon,j,k,npts,
    fitflag = 0 ;
int *countm,*countp,
    *indexm,*indexp ;


long ngoodmag,nbadmag,
     ngoodnav,nbadnav,
     ngoodplat,nbadplat,i ;

long ntot,ncount,ngoodmap,nhead1,nhead2 ;

int *dirflag ;
int insflag[8] ;
int badflag ;

short zone,debug=1 ; /* debug=1 for debug mode */

char filemag[25],filenav[25],fileplat[25],filestd[25] ;
char fileearth[25] ;
char strnset[] = "target\\\\" ;

FILE *fpreadmag,*fpreadnav,*fpreadplat,*fpout,*fpstd ;
FILE *fpearth,*fpdebug ;

Umag *pumag  ; /* pointer to typedef Umag */
Nav  *pnav ; /* pointer to typedef Nav */
Plat *pplat ; /* pointer to typedef Plat */
Map  *pmap ; /* pointer to typedef Map */


/*   Main body      */

/*   Open mag,nav and platform files and output std file */

     puts("Enter name of mag sensor file") ;
     gets(filemag) ;
     puts("Enter name of navigation file") ;
     gets(filenav) ;
     puts("Enter name of platform file") ;
     gets(fileplat) ;
     puts("Enter name of output std file") ;
     gets(filestd) ;

/*   Also open the file that contains the earth mag field inc
```

```
and declination */

puts("Enter name of earth mag parameter file") ;
gets(fileearth) ;


if ((fpreadmag = fopen(filemag,"r")) == NULL)
  {
  fprintf(stderr,"Error opening mag sensor file") ;
  exit(1) ;
  }
if ((fpreadnav = fopen(filenav,"r")) == NULL)
  {
  fprintf(stderr,"Error opening navigation file") ;
  exit(1) ;
  }
if ((fpreadplat = fopen(fileplat,"r")) == NULL)
  {
  fprintf(stderr,"Error opening platform file") ;
  exit(1) ;
  }

if ((fpstd = fopen(filestd,"w")) == NULL)
  {
  fprintf(stderr,"Error opening output std file") ;
  exit(1) ;
  }

if ((fpearth = fopen(fileearth,"r")) == NULL)
  {
  fprintf(stderr,"Earth mag param file not found/using defaults") ;
  }
else
  {
  for (i=0 ; i < 2 ; i++)
    {
    fscanf(fpearth,"%d",&dummy1) ;
      efield[i] = (double)dummy1 ;
    }
  }


/*   Allocate dynamic memory for the Umag,Nav and Plat structures */

if ((pumag = (Umag *)malloc(sizeof(Umag))) == NULL)
  {
  fprintf(stderr,"Allocation for structure Umag failed") ;
  exit(1) ;
  }
if ((pnav = (Nav *)malloc(sizeof(Nav))) == NULL)
  {
```

```
    fprintf(stderr,"Allocation for structure Nav failed") ;
    exit(1) ;
    }
  if ((pplat = (Plat *)malloc(sizeof(Plat))) == NULL)
    {
    fprintf(stderr,"Allocation for structure Plat failed") ;
    exit(1) ;
    }


/*   read data from mag sensor,navigation and platform files */

    puts("\nSTARTED READING IN INPUT STREAMS") ;

    readmag(fpreadmag,pumag,&ngoodmag,&nbadmag) ;
    readnav(fpreadnav,pnav,&ngoodnav,&nbadnav) ;
    readplat(fpreadplat,pplat,&ngoodplat,&nbadplat) ;

    puts("FINISHED READING IN INPUT STREAMS") ;


/*   Allocate dynamic memory for the Map structure */

    if ((pmap = (Map *)malloc(sizeof(Map))) == NULL)
      {
      fprintf(stderr,"Allocation for structure Map failed") ;
      exit(1) ;
      }

/*   Map input to ground based coordinates */

    puts("\nSTARTING MAPPING TO GROUND BASED COORDINATES") ;

    mapper(pumag,pnav,pplat,ngoodmag,ngoodnav,ngoodplat,pmap) ;

    puts("FINISHED MAPPING TO GROUND BASED COORDINATES") ;

/*   Deallocate memory for Umag,Nav and Plat structures */

    free(pumag) ;
    free(pnav) ;
    free(pplat) ;

    ngoodmap = ngoodmag ;
    dirflag = livector(0,ngoodmap) ;
    livector_init(dirflag,ngoodmap,0) ;

    puts("\nSTARTED SENSOR EQUALIZATION") ;

/* Call routine headingindex to get dominant 2 heading directions*/
```

```
        headingindex(pmap,ngoodmap,dirflag,&nhead1,&nhead2) ;

/*  Get median and rms for four mag sensors in both dominant directions */

        getstats(pmap,ngoodmap,dirflag,nhead1,nhead2,insmedian,insrms) ;

/*  Do instrument check using rms for each instrument-return flag field */

        ivector_init(insflag,8,0) ; /* initialize flag field */

        inscheck(insrms,insflag) ;
        badflag = 0 ;
        for (i=0;i<8;i++)
            {
             if (insflag[i] == 1){badflag = 1 ;}
             }
        if (badflag == 1)
             {
             ntot =(3*ngoodmap) ; /* one bad sensor */
             }
        else
             {
             ntot =(4*ngoodmap) ; /* no bad sensors */
             }


/*  Allocate memory for the xdata,ydata,ampdata data arrays */

        xdata = ldvector(0,ntot) ;
        ldvector_init(xdata,ntot,0.) ;



        ydata = ldvector(0,ntot) ;
        ldvector_init(ydata,ntot,0.) ;


        ampdata = ldvector(0,ntot) ;
        ldvector_init(ampdata,ntot,0.) ;


/*  Run routine to take out median and directional offsets and to
    combine into x,y,amp from good sensors into single contiguous arrays */

        combinedata(pmap,ngoodmap,insflag,dirflag,insmedian,insrms,
                xdata,ydata,ampdata,&ncount,&sigma) ;

        puts("FINISHED SENSOR EQUALIZATION") ;

        free_livector(dirflag,0,ngoodmap) ;
```

```
/* Save xy ref and zone from map structure for later use */

      xyref[0] = pmap->xyzero[0] ;
      xyref[1] = pmap->xyzero[1] ;
      zone = pmap->xyzone ;

      free(pmap) ;

      if (debug == 1)
         {
       printf("\nntot=%ld",ntot) ;
       if ((fpdebug = fopen("debug.dat","wb")) == NULL)
        {
        fprintf(stderr,"Error opening debug file") ;
        exit(1) ;
        }
       fwrite(xdata,sizeof(double),ntot,fpdebug) ;
       fwrite(ydata,sizeof(double),ntot,fpdebug) ;
       fwrite(ampdata,sizeof(double),ntot,fpdebug) ;
       }

/*    Allocate dynamic memory initially with maximum allowed data sizes and initialize for arrays to be used in
threshold,cluster algorithms */


     xexceedp = dvector(0,maxexceed) ;
     dvector_init(xexceedp,maxexceed,0.) ;


     yexceedp = dvector(0,maxexceed) ;
     dvector_init(yexceedp,maxexceed,0.) ;


     aexceedp = dvector(0,maxexceed) ;
     dvector_init(aexceedp,maxexceed,0.) ;


     xexceedm = dvector(0,maxexceed) ;
     dvector_init(xexceedm,maxexceed,0.) ;


     yexceedm = dvector(0,maxexceed) ;
     dvector_init(yexceedm,maxexceed,0.) ;


     aexceedm = dvector(0,maxexceed) ;
     dvector_init(aexceedm,maxexceed,0.) ;


     countp = ivector(0,maxcluster) ;
     ivector_init(countp,maxcluster,0) ;
```

```
countm = ivector(0,maxcluster) ;
ivector_init(countm,maxcluster,0) ;


xclusterp = dmatrix(0,maxcluster,0,maxpoints) ;
dmatrix_init(xclusterp,maxcluster,maxpoints,0.) ;


yclusterp = dmatrix(0,maxcluster,0,maxpoints) ;
dmatrix_init(yclusterp,maxcluster,maxpoints,0.) ;


aclusterp = dmatrix(0,maxcluster,0,maxpoints) ;
dmatrix_init(aclusterp,maxcluster,maxpoints,0.) ;

xclusterm = dmatrix(0,maxcluster,0,maxpoints) ;
dmatrix_init(xclusterm,maxcluster,maxpoints,0.) ;

yclusterm = dmatrix(0,maxcluster,0,maxpoints) ;
dmatrix_init(yclusterm,maxcluster,maxpoints,0.) ;

aclusterm = dmatrix(0,maxcluster,0,maxpoints) ;
dmatrix_init(aclusterm,maxcluster,maxpoints,0.) ;


threshplus = 4. * sigma ;
threshminus = -4. * sigma ;


/*   Find the positive data exceedences */

csign = 1 ;
exceed(xdata,ydata,ampdata,ntot,csign,threshplus,xexceedp,yexceedp,aexceedp,&nexceedp) ;

/*   Find the negative data exceedences */

csign = -1 ;
exceed(xdata,ydata,ampdata,ntot,csign,threshminus,xexceedm,yexceedm,aexceedm,&nexceedm) ;


/*   Check if any exceedences-if not don't go any further */

if ((nexceedp == 0) && (nexceedm ==0))
 {
 puts("NO TARGETS FOUND-EXITING PROCESSOR") ;
 fprintf(fpstd,"%s\n",strnset) ;
 free_ldvector(xdata,0,ntot) ;
 free_ldvector(ydata,0,ntot) ;
 free_ldvector(ampdata,0,ntot) ;
 free_dvector(xexceedp,0,maxexceed) ;
 free_dvector(yexceedp,0,maxexceed) ;
```

```
    free_dvector(aexceedp,0,maxexceed) ;
    free_dvector(xexceedm,0,maxexceed) ;
    free_dvector(yexceedm,0,maxexceed) ;
    free_dvector(aexceedm,0,maxexceed) ;
    fclose(fpstd) ;
    exit(1) ;
    }

  puts("\nSTARTED TARGET DETECTION") ;

/*  Find the positive data clusters */

  cluster(nexceedp,xexceedp,yexceedp,aexceedp,&nclusterp,countp,xclusterp,yclusterp,aclusterp) ;

/*  Find the negative data clusters */

  cluster(nexceedm,xexceedm,yexceedm,aexceedm,&nclusterm,countm,xclusterm,yclusterm,aclusterm) ;

/*  Find the centroids of the positive clusters */

  cenxp = dvector(0,nclusterp) ;
  dvector_init(cenxp,nclusterp,0.) ;

  cenyp = dvector(0,nclusterp) ;
  dvector_init(cenyp,nclusterp,0.) ;

  centroid(nclusterp,countp,xclusterp,yclusterp,aclusterp,cenxp,cenyp) ;

/*  Find the centroids of the negative clusters */

  cenxm = dvector(0,nclusterm) ;
  dvector_init(cenxm,nclusterm,0.) ;

  cenym = dvector(0,nclusterm) ;
  dvector_init(cenym,nclusterm,0.) ;

  centroid(nclusterm,countm,xclusterm,yclusterm,aclusterm,cenxm,cenym) ;

  if (debug == 1)
     {
   fwrite(cenxp,sizeof(double),nclusterp,fpdebug) ;
   fwrite(cenyp,sizeof(double),nclusterp,fpdebug) ;
   fwrite(cenxm,sizeof(double),nclusterm,fpdebug) ;
   fwrite(cenym,sizeof(double),nclusterm,fpdebug) ;
   printf("\nnclusterm=%d",nclusterm) ;
   printf("\nnclusterp=%d",nclusterp) ;
    }
```

```
/*   Associate positive and negative clusters together to get dipoles */

     indexp = ivector(0,maxdipoles) ;
     ivector_init(indexp,maxdipoles,-1) ;

     indexm = ivector(0,maxdipoles) ;
     ivector_init(indexm,maxdipoles,-1) ;

     assoc(cenxp,cenyp,cenxm,cenym,nclusterp,nclusterm,&ndipole,indexp,indexm) ;

     puts("COMPLETED TARGET DETECTION") ;

/*   Loop over all dipoles and do dipole fit for each */


     puts("\nSTARTED TARGET CLASSIFICATION") ;

       if (debug == 1)
         {
         printf("\nndipole=%d",ndipole) ;
         debx = dvector(0,ndipole) ;
         dvector_init(debx,ndipole,0.) ;

         deby = dvector(0,ndipole) ;
         dvector_init(deby,ndipole,0.) ;

         fitx = dvector(0,ndipole) ;
         dvector_init(fitx,ndipole,0.) ;

         fity = dvector(0,ndipole) ;
         dvector_init(fity,ndipole,0.) ;

         fitz = dvector(0,ndipole) ;
         dvector_init(fitz,ndipole,0.) ;

         fiti = dvector(0,ndipole) ;
         dvector_init(fiti,ndipole,0.) ;

         fitd = dvector(0,ndipole) ;
         dvector_init(fitd,ndipole,0.) ;

         fitamp = dvector(0,ndipole) ;
         dvector_init(fitamp,ndipole,0.) ;

         fitchi = dvector(0,ndipole) ;
         dvector_init(fitchi,ndipole,0.) ;

         }
```

```
      for (i=0; i < ndipole ;i++)
        {
/*      get starting parameter values for later dipole fit */
        initparams(xclusterm,yclusterm,xclusterp,yclusterp,aclusterm,aclusterp,
                cenxp,cenyp,cenxm,cenym,indexm,indexp,i,countp,countm,initparam) ;

        if (debug == 1)
          {
          debx[i] = initparam[0] ;
          deby[i] = initparam[1] ;
          }

/*    concatenate positive and negative clusters into one vector */

        nsize = 2*maxpoints ;
        xcon = dvector(0,nsize) ;
        dvector_init(xcon,nsize,0.) ;

        ycon = dvector(0,nsize) ;
        dvector_init(ycon,nsize,0.) ;

        ampcon = dvector(0,nsize) ;
        dvector_init(ampcon,nsize,0.) ;

        prepdipole(xclusterm,yclusterm,xclusterp,yclusterp,aclusterm,aclusterp,
                indexm,indexp,i,countp,countm,xcon,ycon,ampcon,&nptcon) ;

        if (nptcon < ncrit) /* not enough fit points-expand region and trim to actual size */
           {
          expand(initparam,xdata,ydata,ampdata,ntot,xcon,ycon,ampcon,&nptcon) ;

          xf = dvector(0,nptcon) ;
          dvector_init(xf,nptcon,0.) ;

          yf = dvector(0,nptcon) ;
          dvector_init(yf,nptcon,0.) ;

          af = dvector(0,nptcon) ;
          dvector_init(af,nptcon,0.) ;

          trim(xcon,ycon,ampcon,nptcon,xf,yf,af) ;
          free_dvector(xcon,0,nsize) ;
          free_dvector(ycon,0,nsize) ;
          free_dvector(ampcon,0,nsize) ;
          npts = nptcon ;
          }
        else /* trim arrays down to actual size for input to dipole routine */
           {
          xf = dvector(0,nptcon) ;
          dvector_init(xf,nptcon,0.) ;
```

```
            yf = dvector(0,nptcon) ;
            dvector_init(yf,nptcon,0.) ;

            af = dvector(0,nptcon) ;
            dvector_init(af,nptcon,0.) ;

            trim(xcon,ycon,ampcon,nptcon,xf,yf,af) ;
            free_dvector(xcon,0,nsize) ;
            free_dvector(ycon,0,nsize) ;
            free_dvector(ampcon,0,nsize) ;
            npts = nptcon ;
              }
          dipole(npts,xf,yf,af,sigma,efield,initparam,param,error,&amp,&amperr,&chisq,&fitflag) ; /* does dipole
 shape fit and then amp fit */

          if (debug == 1)
            {
            fitx[i] = param[0] ;
            fity[i] = param[1] ;
            fitz[i] = param[2] ;
            fiti[i] = param[3] ;
            fitd[i] = param[4] ;
            fitamp[i] = amp ;
            fitchi[i] = chisq ;
              }

          if (fitflag == 0)
            {
/*      Write out results to stdfile */
            outstd(fpstd,zone,fitflag,i,param,error,amp,amperr,chisq,xyref) ;
            }
          else

/* Write out results to stdfile */

            {
            amp = 99999. ; /* default setting */
            for (k =0; k < 5 ; k++)
              {
              param[k] = initparam[k] ;
              }

/*      Write out results to stdfile */

            outstd(fpstd,zone,fitflag,i,param,error,amp,amperr,chisq,xyref) ;

            }

          free_dvector(xf,0,npts) ;
          free_dvector(yf,0,npts) ;
          free_dvector(af,0,npts) ;
```

```
        }

      puts("COMPLETED TARGET CLASSIFICATION") ;

      fclose(fpstd) ; /* close the output file */
      if (debug == 1)
        {
        fwrite(debx,sizeof(double),ndipole,fpdebug) ;
        fwrite(deby,sizeof(double),ndipole,fpdebug) ;
        fwrite(fitx,sizeof(double),ndipole,fpdebug) ;
        fwrite(fity,sizeof(double),ndipole,fpdebug) ;
        fwrite(fitz,sizeof(double),ndipole,fpdebug) ;
        fwrite(fiti,sizeof(double),ndipole,fpdebug) ;
        fwrite(fitd,sizeof(double),ndipole,fpdebug) ;
        fwrite(fitamp,sizeof(double),ndipole,fpdebug) ;
        fwrite(fitchi,sizeof(double),ndipole,fpdebug) ;
        }


      if (debug ==1){fclose(fpdebug) ;}

/*  deallocate dynamic memory */

      free_ldvector(xdata,0,ntot) ;
      free_ldvector(ydata,0,ntot) ;
      free_ldvector(ampdata,0,ntot) ;
      free_dvector(xexceedp,0,maxexceed) ;
      free_dvector(yexceedp,0,maxexceed) ;
      free_dvector(aexceedp,0,maxexceed) ;
      free_dvector(xexceedm,0,maxexceed) ;
      free_dvector(yexceedm,0,maxexceed) ;
      free_dvector(aexceedm,0,maxexceed) ;
      free_ivector(countp,0,maxcluster) ;
      free_ivector(countm,0,maxcluster) ;
      free_dvector(cenxp,0,nclusterp) ;
      free_dvector(cenyp,0,nclusterp) ;
      free_dvector(cenxm,0,nclusterm) ;
      free_dvector(cenym,0,nclusterm) ;
      free_ivector(indexp,0,maxdipoles) ;
      free_ivector(indexm,0,maxdipoles) ;
      free_dmatrix(xclusterp,0,maxcluster,0,maxpoints) ;
      free_dmatrix(yclusterp,0,maxcluster,0,maxpoints) ;
      free_dmatrix(aclusterp,0,maxcluster,0,maxpoints) ;
      free_dmatrix(xclusterm,0,maxcluster,0,maxpoints) ;
      free_dmatrix(yclusterm,0,maxcluster,0,maxpoints) ;
      free_dmatrix(aclusterm,0,maxcluster,0,maxpoints) ;

}
```

```
/*****************************************************************************
*
*    Routine: mapper.c
*
*    This module takes data from the Umag,Nav and Plat structures
*    and maps to ground-based coordinates for each sensor.
*
*    Doug DeProspo
*    14Jun95
*
*        Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include "dipole.h"
#define MTF 3.280833

void mapper(Umag *pumag,Nav *pnav,Plat *pplat,long ngoodmag,long ngoodnav,long ngoodplat,Map *pmap)



{
     double plattime,navtime,magtime,shiftnav,shiftplat ;
     double time1,time2,deltat,dt1,dt2,weight1,weight2,xtract,ytract ;
     double dx[4],dy[4] ;
     double dummy1,dummy2,dummy3,dummy4 ;
     double dummy5,dummy6,dummy7,dummy8 ;
     double dummy9,dummy10,dummy11 ;
     long iplatlow=0,inavlow=0 ;
     long navindex1,navindex2,platindex ;
     long i ;


     shiftnav = (pumag->tzeromag)-(pnav->tzeronav) ;
     shiftplat = (pumag->tzeromag)-(pplat->tzeroplat) ;
     pmap->xyzero[0] = (pnav->xyzero[0]) * MTF ; /* convert to ft */
     pmap->xyzero[1] = (pnav->xyzero[1]) * MTF ;
     pmap->tzeromap = pumag->tzeromag ;
     pmap->xyzone = pnav->llzone ;
      dummy1 = pmap->xyzero[0] ;
      dummy2 = pmap->xyzero[1] ;
      dummy3 = pmap->tzeromap ;
     for (i=0; i<ngoodmag ; i++)
        {
      magtime = pumag->dtmag[i] ;
      navtime = magtime + shiftnav ;
      plattime = magtime + shiftplat ;

/*  Call routine to get indices for nav records nearest to mag time */
```

```
        getxytractor(navtime,pnav,ngoodnav,inavlow,&navindex1,&navindex2) ;

        if (navindex1 == -1)
          {
          navindex1 = inavlow ;
          navindex2 = inavlow ;
          }

        inavlow = navindex1 ;
        time1 = pnav->dtnav[navindex1] ;
        time2 = pnav->dtnav[navindex2] ;
        deltat = time2 - time1 ;
        if (deltat != 0.)
          {
          dt1 = fabs(navtime - time1) ;
          dt2 = fabs(navtime - time2) ;
          weight1 = dt1/deltat ;
          weight2 = dt2/deltat ;
            }
        else
          {
          weight1 = .5 ;
          weight2 = .5 ;
            }

/*  Linearly interpolate between  values at 2 endpoints */

        xtract = (weight1*(pnav->xtractor[navindex1])
                + weight2*(pnav->xtractor[navindex2]))/(weight1+weight2) ;
        ytract = (weight1*(pnav->ytractor[navindex1])
                + weight2*(pnav->ytractor[navindex2]))/(weight1+weight2) ;

/*  Convert to ft. */

        xtract *= MTF ;
        ytract *= MTF ;

/*  Call routine to get dx,dy offsets for mags */

        getdxymags(plattime,pplat,ngoodplat,iplatlow,&platindex,dx,dy) ;

        iplatlow = platindex ;

/*  Load output structure Map */

        pmap->dtmap[i] = pumag->dtmag[i] ;
        pmap->heading[i] = pplat->heading[platindex] ;
        pmap->amp[0][i] = pumag->ampa[i] ;
        pmap->amp[1][i] = pumag->ampb[i] ;
        pmap->amp[2][i] = pumag->ampc[i] ;
        pmap->amp[3][i] = pumag->ampd[i] ;
```

```
            pmap->xmap[0][i] = xtract + dx[0] ;
            pmap->xmap[1][i] = xtract + dx[1] ;
            pmap->xmap[2][i] = xtract + dx[2] ;
            pmap->xmap[3][i] = xtract + dx[3] ;
            pmap->ymap[0][i] = ytract + dy[0] ;
            pmap->ymap[1][i] = ytract + dy[1] ;
            pmap->ymap[2][i] = ytract + dy[2] ;
            pmap->ymap[3][i] = ytract + dy[3] ;


            dummy4 = pmap->amp[1][i] ;
            dummy5 = pmap->amp[2][i] ;
            dummy6 = pmap->amp[3][i] ;
            dummy8 = pmap->xmap[0][i] ;
            dummy9 = pmap->xmap[1][i] ;
            dummy10 = pmap->xmap[2][i] ;
            dummy11 = pmap->xmap[3][i] ;

            }

        return ;

}
```

```
/*************************nrutil.c*************************************/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "nrutil.h"

void nrerror(char *error_text)
{
        void exit( int i);

        fprintf(stderr,"Numerical Recipes run-time error...\n");
        fprintf(stderr,"%s\n",error_text);
        fprintf(stderr,"...now exiting to system...\n");
        exit(1);
}



float *vector( int nl, int nh)
{
        float *v;

        v=(float *)malloc((unsigned) (nh-nl+1)*sizeof(float));
        if (!v) nrerror("allocation failure in vector()");
        return v-nl;
}

int *ivector( int nl, int nh)
{
        int *v;

        v=(int *)malloc((unsigned) (nh-nl+1)*sizeof(int));
        if (!v) nrerror("allocation failure in ivector()");
        return v-nl;
}
int *livector( int nl, long nh)
{
        int *v;

        v=(int *)malloc((unsigned) (nh-nl+1)*sizeof(int));
        if (!v) nrerror("allocation failure in ivector()");
        return v-nl;
}

double *dvector( int nl, int nh)
{
        double *v;

        v=(double *)malloc((unsigned) (nh-nl+1)*sizeof(double));
        if (!v) nrerror("allocation failure in dvector()");
        return v-nl;
```

```
}
double *ldvector( int nl, long nh)
{
        double *v;

        v=(double *)malloc((unsigned) (nh-nl+1)*sizeof(double));
        if (!v) nrerror("allocation failure in dvector()");
        return v-nl;
}



float **matrix( int nrl, int nrh, int ncl, int nch)
{
        int i;
        float **m;

        m=(float **) malloc((unsigned) (nrh-nrl+1)*sizeof(float*));
        if (!m) nrerror("allocation failure 1 in matrix()");
        m -= nrl;

        for(i=nrl;i<=nrh;i++) {
                m[i]=(float *) malloc((unsigned) (nch-ncl+1)*sizeof(float));
                if (!m[i]) nrerror("allocation failure 2 in matrix()");
                m[i] -= ncl;
        }
        return m;
}

double **dmatrix( int nrl, int nrh, int ncl, int nch)
{
        int i;
        double **m;

        m=(double **) malloc((unsigned) (nrh-nrl+1)*sizeof(double*));
        if (!m) nrerror("allocation failure 1 in dmatrix()");
        m -= nrl;

        for(i=nrl;i<=nrh;i++) {
                m[i]=(double *) malloc((unsigned) (nch-ncl+1)*sizeof(double));
                if (!m[i]) nrerror("allocation failure 2 in dmatrix()");
                m[i] -= ncl;
        }
        return m;
}

int **imatrix( int nrl, int nrh, int ncl, int nch)
{
        int i,**m;

        m=(int **)malloc((unsigned) (nrh-nrl+1)*sizeof(int*));
```

```
        if (!m) nrerror("allocation failure 1 in imatrix()");
        m -= nrl;

        for(i=nrl;i<=nrh;i++) {
                m[i]=(int *)malloc((unsigned) (nch-ncl+1)*sizeof(int));
                if (!m[i]) nrerror("allocation failure 2 in imatrix()");
                m[i] -= ncl;
        }
        return m;
}


short int **hmatrix( int nrl, int nrh, int ncl, int nch)
{
        int i;
        short int **m;

        m=(short int **)malloc((unsigned) (nrh-nrl+1)*sizeof(short int*));
        if (!m) nrerror("allocation failure 1 in hmatrix()");
        m -= nrl;

        for(i=nrl;i<=nrh;i++) {
                m[i]=
                  (short int *)malloc((unsigned) (nch-ncl+1)*sizeof(short int));
                if (!m[i]) nrerror("allocation failure 2 in hmatrix()");
                m[i] -= ncl;
        }
        return m;
}


float **submatrix( float **a, int oldrl, int oldrh, int oldcl, int oldch,
  int newrl, int newcl)
{
        int i,j;
        float **m;

        m=(float **) malloc((unsigned) (oldrh-oldrl+1)*sizeof(float*));
        if (!m) nrerror("allocation failure in submatrix()");
        m -= newrl;

        for(i=oldrl,j=newrl;i<=oldrh;i++,j++) m[j]=a[i]+oldcl-newcl;

        return m;
}



void free_vector( float *v, int nl, int nh)
{
        free((char*) (v+nl));
```

```
}

void free_ivector( int *v, int nl, int nh)
{
        free((char*) (v+nl));
}
void free_livector( int *v, int nl, long nh)
{
        free((char*) (v+nl));
}

void free_dvector( double *v, int nl, int nh)
{
        free((char*) (v+nl));
}
void free_ldvector( double *v, int nl, long nh)
{
        free((char*) (v+nl));
}



void free_matrix( float **m, int nrl, int nrh, int ncl, int nch)
{
        int i;

        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}

void free_dmatrix( double **m, int nrl, int nrh, int ncl, int nch)
{
        int i;

        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}

void free_imatrix( int **m, int nrl, int nrh, int ncl, int nch)
{
        int i;

        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}

void free_hmatrix( short int **m, int nrl, int nrh, int ncl, int nch)
{
        int i;

        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
```

```
        free((char*) (m+nrl));
}


void free_submatrix( float **b, int nrl, int nrh, int ncl, int nch)
{
        free((char*) (b+nrl));
}


float **convert_matrix( float *a, int nrl, int nrh, int ncl, int nch)
{
        int i,j,nrow,ncol;
        float **m;

        nrow=nrh-nrl+1;
        ncol=nch-ncl+1;
        m = (float **) malloc((unsigned) (nrow)*sizeof(float*));
        if (!m) nrerror("allocation failure in convert_matrix()");
        m -= nrl;
        for(i=0,j=nrl;i<=nrow-1;i++,j++) m[j]=a+ncol*i-ncl;
        return m;
}


void free_convert_matrix( float **b, int nrl, int nrh, int ncl, int nch)
{
        free((char*) (b+nrl));
}
```

```
/*******************************nrutil.h************************************/
void nrerror(char *error_text);
float *vector(int nl, int nh );
int *ivector(  int nl, int nh);
double *dvector( int nl, int nh );
float **matrix( int nrl, int nrh, int ncl, int nch);
double **dmatrix( int nrl, int nrh, int ncl, int nch );
int **imatrix(int nrl, int nrh, int ncl, int nch );
short int **hmatrix( int nrl, int nrh, int ncl, int nch);
float **submatrix( float **a, int oldrl, int oldrh, int oldcl, int oldch, int newrl, int newcl);
void free_vector(float *v, int nl, int nh );
void free_ivector(int *v, int nl, int nh );
void free_dvector( double *v, int nl, int nh);
void free_matrix(float **m, int nrl, int nrh, int ncl, int nch );
void free_dmatrix(double **m, int nrl, int nrh, int ncl, int nch );
void free_imatrix(int **m, int nrl, int nrh, int ncl, int nch );
void free_hmatrix( short int **m, int nrl, int nrh, int ncl, int nch);
void free_submatrix(float **b, int nrl, int nrh, int ncl, int nch );
float **convert_matrix( float *a, int nrl, int nrh, int ncl, int nch);
void free_convert_matrix( float **b , int nrl, int nrh, int ncl, int nch);
```

```
/*****************************************************************************
*
*     Routine: outstd.c
*
*     This module outputs into the std file according
*     to a prescribed format in ...
*
*     Doug DeProspo
*     30Jun95
*
*          Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include <string.h>
#include "dipole.h"
#define FTM .304800
#define starget .485
#define btarget .917


      double latdeg,londeg,latmin,lonmin,lat,lon ;
      double xloc,yloc,conf,size,sizeerr,bound[2] ;
      char strtset[] = "target\\\\" ;
      char str1[] = "stdRec" ;
      char str2[] = "targetLocation " ;
      char str3[] = "targetDepth " ;
      char str4[] = "targetSize " ;
      char str5[] = "sizeConfidence " ;
      char strbad[] = "*\\" ;
      char ssizes[] = "S" ;
      char ssizem[] = "M" ;
      char ssizel[] = "L" ;

void outstd(FILE *fpstd,short zone,int fitflag,int i,double param[],double error[],
          double amp,double amperr,double chisq,double xyref[2])



{

     bound[0] = (double)starget ;
     bound[1] = (double)btarget ;

     xloc = xyref[0] + param[0] ;
     yloc = xyref[1] + param[1] ;

     xloc *= FTM ;
     yloc *= FTM ;

     contoll(xloc,yloc,zone,&lat,&lon) ;
```

```
/*  Convert lat/long which is in degrees and fractions of a degree to
    degrees/minutes/fractions of minute(5 decimal places) */

    lon = fabs(lon) ;

    conmin(lat,&latdeg,&latmin) ;
    conmin(lon,&londeg,&lonmin) ;

     if (i == 0)
       {
       fprintf(fpstd,"%s\n",strtset) ; /*  Write target set delimiter */
       }

/*  Irrespective of flag write out record field and location(lat/long) */

     fprintf(fpstd,"%s\\\n",str1) ;
     fprintf(fpstd,"%s%2d%7.5f:%3d%8.5f\\\n",str2,
           (int)latdeg,latmin,(int)londeg,
           lonmin) ;

     if (fitflag == 0) /* A good fit */
       {
       fprintf(fpstd,"%s%.2f\\\n",str3,-1.*param[2]*FTM) ; /* target depth */
       fprintf(fpstd,"%s\n",strbad) ; /* no target phi */
       fprintf(fpstd,"%s\n",strbad) ; /* no target theta */

/* target size */

       size = 2. * amp ;
       sizeerr = 2. * amperr ;
       if (size <= bound[0]){fprintf(fpstd,"%s%s\\\n",str4,ssizes) ;}
       if (size >= bound[1]){fprintf(fpstd,"%s%s\\\n",str4,ssizel) ;}
       if ((size > bound[0]) && (size < bound[1]))
               {fprintf(fpstd,"%s%s\\\n",str4,ssizem) ;}

       fprintf(fpstd,"%s\n",strbad) ; /* no target length */
       fprintf(fpstd,"%s\n",strbad) ; /* no target type */

       getconf(size,sizeerr,bound,&conf) ;

       fprintf(fpstd,"%s%3d\\\n",str5,(int)conf) ; /* target size confidence */
       fprintf(fpstd,"%s\n",strbad) ; /* no target type confidence */

       }
     else /* A bad fit */
       {
       fprintf(fpstd,"%s\n",strbad) ; /* no target depth */
       fprintf(fpstd,"%s\n",strbad) ; /* no target phi */
       fprintf(fpstd,"%s\n",strbad) ; /* no target theta */
       fprintf(fpstd,"%s\n",strbad) ; /* no target size */
       fprintf(fpstd,"%s\n",strbad) ; /* no target length */
```

```
        fprintf(fpstd,"%s\n",strbad) ; /* no target type */
        fprintf(fpstd,"%s\n",strbad) ; /* no target size confidence */
        fprintf(fpstd,"%s\n",strbad) ; /* no target type confidence */
        }

    return ;

}
```

```
/*****************************************************************************
*
*    Routine: prepdipole.c
*
*    This module takes the + and - clusters of a dipole(if 2-pole)
*    and concatenates them into contiguous vectors.
*
*    Doug DeProspo
*    22Dec94
*        Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include "dipole.h"


void prepdipole(double **xclusterm,double **yclusterm,double **xclusterp,double **yclusterp,
            double **aclusterm,double **aclusterp,int *indexm,int *indexp,int ndip,
            int *countp,int *countm,double *xf,double *yf,double *af,int *npts)



{

    int mpts,i,offset ;

    if ((indexp[ndip] != -1) && (indexm[ndip] != -1))
/*  2 pole dipole */
      {
      mpts = countp[indexp[ndip]] + countm[indexm[ndip]] ;
      for (i=0;i < countp[indexp[ndip]];i++)
          {
        xf[i] = xclusterp[indexp[ndip]][i] ;
        yf[i] = yclusterp[indexp[ndip]][i] ;
        af[i] = aclusterp[indexp[ndip]][i] ;
          }
      for (i=0; i < countm[indexm[ndip]];i++)
          {
        offset = countp[indexp[ndip]] ;
        xf[i+offset] = xclusterm[indexm[ndip]][i] ;
        yf[i+offset] = yclusterm[indexm[ndip]][i] ;
        af[i+offset] = aclusterm[indexm[ndip]][i] ;
        }
      }
    else
      {
      if (indexm[ndip] == -1)
/*    single + pole  */
        {
        mpts = countp[indexp[ndip]] ;
        for (i=0;i < mpts;i++)
```

```
            {
          xf[i] = xclusterp[indexp[ndip]][i] ;
          yf[i] = yclusterp[indexp[ndip]][i] ;
          af[i] = aclusterp[indexp[ndip]][i] ;
            }
        }
      else
/*    single - pole  */
        {
        mpts = countm[indexm[ndip]] ;
        for (i=0;i < mpts;i++)
            {
          xf[i] = xclusterm[indexm[ndip]][i] ;
          yf[i] = yclusterm[indexm[ndip]][i] ;
          af[i] = aclusterm[indexm[ndip]][i] ;
            }
        }
      }

    *npts = mpts ;
    return ;
}
```

```
/*****************************************************************************
*
*     Routine: readmag.c
*
*     This module reads in mag sensor data into the structure
*     Umag which is defined in dipole.h
*
*     Doug DeProspo
*     26April95
*          Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include <string.h>
#include "dipole.h"


#define BUFLENSENSOR 90 /* largest number of characters allowed in line */
#define CHECKPOS 82 /* expected position offset of the // delimeters */
#define OFFSET_1 14 /* offset of julian hour */
#define OFFSET_2 17 /* offset of julian min */
#define OFFSET_3 20 /* offset of julian sec */
#define OFFSET_4 39 /* offset of sensor a */
#define OFFSET_5 50 /* offset of sensor b */
#define OFFSET_6 61 /* offset of sensor c */
#define OFFSET_7 72 /* offset of sensor d */
#define SIZE_1 2 /* size of julian hour */
#define SIZE_2 2 /* size of julian min */
#define SIZE_3 9 /* size of julian sec */
#define SIZE_4 10 /* size of sensor a */
#define SIZE_5 10 /* size of sensor b */
#define SIZE_6 10 /* size of sensor c */
#define SIZE_7 10 /* size of sensor d */


void readmag(FILE *fpreadmag,Umag *pumag,long *ngoodmag,
          long *nbadmag)


{
    char buf[BUFLENSENSOR] ;
    char match[] = "\\\\" ;
    char *st_julian_hour,*st_julian_min,*st_julian_sec,
        *st_ampa,*st_ampb,*st_ampc,*st_ampd ;
    char *loc ;

    double julian_hour,julian_min,julian_sec ;
    double time,time0 ;
    long mgoodmag=0,mbadmag=0 ;

/*  allocate memory for substrings */
```

```
        st_julian_hour = (char *)malloc(SIZE_1+1) ;
        st_julian_min = (char *)malloc(SIZE_2+1) ;
        st_julian_sec = (char *)malloc(SIZE_3+1) ;
        st_ampa = (char *)malloc(SIZE_4+1) ;
        st_ampb = (char *)malloc(SIZE_5+1) ;
        st_ampc = (char *)malloc(SIZE_6+1) ;
        st_ampd = (char *)malloc(SIZE_7+1) ;

    /*  read in data line by line */
        while(!feof(fpreadmag))
          {
          fgets(buf,BUFLENSENSOR,fpreadmag) ;
          loc = strstr(buf,match) ;
          if (loc == NULL)
            {
            mbadmag += 1 ;
            }
          else
            {
            if ((loc-buf) != CHECKPOS)
              {
              mbadmag += 1 ;
              }
            else
              {
              strncpy(st_julian_hour,buf+OFFSET_1,SIZE_1) ;
              strncpy(st_julian_min,buf+OFFSET_2,SIZE_2) ;
              strncpy(st_julian_sec,buf+OFFSET_3,SIZE_3) ;
              strncpy(st_ampa,buf+OFFSET_4,SIZE_4) ;
              strncpy(st_ampb,buf+OFFSET_5,SIZE_5) ;
              strncpy(st_ampc,buf+OFFSET_6,SIZE_6) ;
              strncpy(st_ampd,buf+OFFSET_7,SIZE_7) ;

    /*  put in null terminators for sub-strings */

              st_julian_hour[SIZE_1] = '\0' ;
              st_julian_min[SIZE_2] = '\0' ;
              st_julian_sec[SIZE_3] = '\0' ;
              st_ampa[SIZE_4] = '\0' ;
              st_ampb[SIZE_5] = '\0' ;
              st_ampc[SIZE_6] = '\0' ;
              st_ampd[SIZE_7] = '\0' ;

    /*  convert strings to floating point numbers */

              julian_hour = atof(st_julian_hour) ;
              julian_min = atof(st_julian_min) ;
              julian_sec = atof(st_julian_sec) ;
              time = julian_hour*3600. + julian_min*60. + julian_sec ;

              if (mgoodmag == 0)
```

```
            {
            time0 = time ;
            pumag->dtmag[mgoodmag] = 0. ;
            pumag->tzeromag = time ;
            }
          else
            {
            pumag->dtmag[mgoodmag] = time-time0 ;
            }
          pumag->ampa[mgoodmag] = atof(st_ampa)/100000. ;
          pumag->ampb[mgoodmag] = atof(st_ampb)/100000. ;
          pumag->ampc[mgoodmag] = atof(st_ampc)/100000. ;
          pumag->ampd[mgoodmag] = atof(st_ampd)/100000. ;
          mgoodmag += 1 ;

          }
        }
      }

    *ngoodmag = mgoodmag ;
    *nbadmag = mbadmag ;

    free(st_julian_hour) ;
    free(st_julian_min) ;
    free(st_julian_sec) ;
    free(st_ampa) ;
    free(st_ampb) ;
    free(st_ampc) ;
    free(st_ampd) ;

    fclose(fpreadmag) ;
    return ;

}
```

```
/*****************************************************************************
*
*     Routine: readnav.c
*
*     This module reads the navigation data into the structure
*     Nav which is typedef in dipole.h
*
*     Doug DeProspo
*     15May95
*     15Jun95 : Modified to include lat/long to xy conversion subroutine
*
*          Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include <string.h>
#include "dipole.h"


#define BUFLENSENSOR 72 /* largest number of characters allowed in line */
#define CHECKPOS 66 /* expected position offset of the // delimeters */
#define OFFSET_1 11 /* offset of julian hour */
#define OFFSET_2 14 /* offset of julian min */
#define OFFSET_3 17 /* offset of julian sec */
#define OFFSET_4 27 /* offset of lattitude */
#define OFFSET_5 39 /* offset of longitude */
#define OFFSET_6 51 /* offset of status */
#define OFFSET_7 59 /* offset of altitude */
#define SIZE_1 2 /* size of julian hour */
#define SIZE_2 2 /* size of julian min */
#define SIZE_3 9 /* size of julian sec */
#define SIZE_4 11 /* size of lattitude */
#define SIZE_5 11 /* size of longitude */
#define SIZE_6 7 /* size of status */
#define SIZE_7 7 /* size of altitude */

void readnav(FILE *fpreadnav,Nav *punav,long *ngoodnav,
          long *nbadnav)


{
    char buf[BUFLENSENSOR] ;
    char match[] = "\\\\" ;
    char *st_julian_hour,*st_julian_min,*st_julian_sec,
        *st_lat,*st_long,*st_stat,*st_alt ;
    char *loc ;
    double julian_hour,julian_min,julian_sec ;
    double time,time0 ;
    double lattitude,longitude ;
    double x,y,x0,y0,xprev,yprev,cdist ;
    long mgoodnav=0,mbadnav=0 ;
    short zone ;
```

```
/*  allocate memory for substrings */

    st_julian_hour = (char *)malloc(SIZE_1+1) ;
    st_julian_min = (char *)malloc(SIZE_2+1) ;
    st_julian_sec = (char *)malloc(SIZE_3+1) ;
    st_lat = (char *)malloc(SIZE_4+1) ;
    st_long = (char *)malloc(SIZE_5+1) ;
    st_stat = (char *)malloc(SIZE_6+1) ;
    st_alt = (char *)malloc(SIZE_7+1) ;

/*  read in data line by line */
    while(!feof(fpreadnav))
     {
     fgets(buf,BUFLENSENSOR,fpreadnav) ;
     loc = strstr(buf,match) ;
     if (loc == NULL)
       {
       mbadnav += 1 ;
       }
     else
       {
       if ((loc-buf) != CHECKPOS)
         {
         mbadnav += 1 ;
         }
       else
         {
         strncpy(st_julian_hour,buf+OFFSET_1,SIZE_1) ;
         strncpy(st_julian_min,buf+OFFSET_2,SIZE_2) ;
         strncpy(st_julian_sec,buf+OFFSET_3,SIZE_3) ;
         strncpy(st_lat,buf+OFFSET_4,SIZE_4) ;
         strncpy(st_long,buf+OFFSET_5,SIZE_5) ;
         strncpy(st_stat,buf+OFFSET_6,SIZE_6) ;
         strncpy(st_alt,buf+OFFSET_7,SIZE_7) ;

/*  put in null terminators for sub-strings */

         st_julian_hour[SIZE_1] = '\0' ;
         st_julian_min[SIZE_2] = '\0' ;
         st_julian_sec[SIZE_3] = '\0' ;
         st_lat[SIZE_4] = '\0' ;
         st_long[SIZE_5] = '\0' ;
         st_stat[SIZE_6] = '\0' ;
         st_alt[SIZE_7] = '\0' ;

/*  convert strings to floating point numbers */

         julian_hour = atof(st_julian_hour) ;
         julian_min = atof(st_julian_min) ;
         julian_sec = atof(st_julian_sec) ;
```

```
        time = julian_hour*3600. + julian_min*60. + julian_sec ;
        lattitude = atof(st_lat ) ;
        longitude = -1. * atof(st_long ) ;

/*  call routine to convert lat/long to utm x,y */

        if (mgoodnav == 0){getzone(-1.*longitude,&zone) ; }


        contoxy(lattitude,longitude,zone,&x,&y) ;

        if (mgoodnav == 0)
          {
          time0 = time ;
          punav->dtnav[0] = 0. ;
          punav->llzone = zone ;
          punav->tzeronav = time ;
          x0 = x ;
          y0 = y ;
          punav->xyzero[0] = x0 ;
          punav->xyzero[1] = y0 ;
          punav->xtractor[0] = 0. ;
          punav->ytractor[0] = 0. ;
          punav->navstat[mgoodnav] = atof(st_stat ) ;
          punav->altitude[mgoodnav] = atof(st_alt ) ;
          mgoodnav += 1 ;
          xprev = 0. ;
          yprev = 0. ;
            }
        else
          {
          cdist = sqrt(pow(((x-x0)-xprev),2.) + pow(((y-y0)-yprev),2.)) ;
          if (cdist != 0.)
            {
            punav->dtnav[mgoodnav] = time-time0 ;
            punav->xtractor[mgoodnav] = x - x0 ;
            punav->ytractor[mgoodnav] = y - y0 ;
            punav->navstat[mgoodnav] = atof(st_stat ) ;
            punav->altitude[mgoodnav] = atof(st_alt ) ;
            xprev = x - x0 ;
            yprev = y - y0 ;
            mgoodnav += 1 ;
              }
          else
            {
            xprev = x - x0 ;
            yprev = y - y0 ;
            mbadnav += 1 ;
            }
          }
        }
```

```
        }
      }


    *ngoodnav = mgoodnav ;
    *nbadnav = mbadnav ;

    free(st_julian_hour) ;
    free(st_julian_min) ;
    free(st_julian_sec) ;
    free(st_lat) ;
    free(st_long) ;
    free(st_stat) ;
    free(st_alt) ;

    fclose(fpreadnav) ;
    return ;

}
```

```
/*****************************************************************************
*
*     Routine: readplat.c
*
*     This module reads in platform data into the structure
*     Plat which is defined in dipole.h
*
*     Doug DeProspo
*     15May95
*          Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include <string.h>
#include "dipole.h"


#define BUFLENSENSOR 94 /* largest number of characters allowed in line */
#define CHECKPOS 86 /* expected position offset of the // delimeters */
#define OFFSET_1 16 /* offset of julian hour */
#define OFFSET_2 19 /* offset of julian min */
#define OFFSET_3 22 /* offset of julian sec */
#define OFFSET_4 32 /* offset of heading */
#define OFFSET_5 39 /* offset of vpitch */
#define OFFSET_6 47 /* offset of vroll */
#define OFFSET_7 55 /* offset of hpitch */
#define OFFSET_8 63 /* offset of hroll */
#define OFFSET_9 71 /* offset of hyaw */
#define SIZE_1 2 /* size of julian hour */
#define SIZE_2 2 /* size of julian min */
#define SIZE_3 9 /* size of julian sec */
#define SIZE_4 6 /* size of heading */
#define SIZE_5 7 /* size of vpitch */
#define SIZE_6 7 /* size of vroll */
#define SIZE_7 7 /* size of hpitch */
#define SIZE_8 7 /* size of hroll */
#define SIZE_9 7 /* size of hyaw */


void readplat(FILE *fpreadplat,Plat *puplat,long *ngoodplat,
          long *nbadplat)


{
    char buf[BUFLENSENSOR] ;
    char match[] = "\\\\" ;
    char *st_julian_hour,*st_julian_min,*st_julian_sec,
        *st_heading,*st_vpitch,*st_vroll,*st_hpitch,
        *st_hroll,*st_hyaw ;
    char *loc ;
```

```
    double julian_hour,julian_min,julian_sec ;
    double time,time0 ;
    long mgoodplat=0,mbadplat=0 ;

/*  allocate memory for substrings */

    st_julian_hour = (char *)malloc(SIZE_1+1) ;
    st_julian_min = (char *)malloc(SIZE_2+1) ;
    st_julian_sec = (char *)malloc(SIZE_3+1) ;
    st_heading = (char *)malloc(SIZE_4+1) ;
    st_vpitch = (char *)malloc(SIZE_5+1) ;
    st_vroll = (char *)malloc(SIZE_6+1) ;
    st_hpitch = (char *)malloc(SIZE_7+1) ;
    st_hroll = (char *)malloc(SIZE_8+1) ;
    st_hyaw = (char *)malloc(SIZE_9+1) ;


/*  read in data line by line */
    while(!feof(fpreadplat))
     {
     fgets(buf,BUFLENSENSOR,fpreadplat) ;
     loc = strstr(buf,match) ;
     if (loc == NULL)
       {
       mbadplat += 1 ;
       }
      else
       {
       if ((loc-buf) != CHECKPOS)
         {
         mbadplat += 1 ;
         }
        else
         {
         strncpy(st_julian_hour,buf+OFFSET_1,SIZE_1) ;
         strncpy(st_julian_min,buf+OFFSET_2,SIZE_2) ;
         strncpy(st_julian_sec,buf+OFFSET_3,SIZE_3) ;
         strncpy(st_heading,buf+OFFSET_4,SIZE_4) ;
         strncpy(st_vpitch,buf+OFFSET_5,SIZE_5) ;
         strncpy(st_vroll,buf+OFFSET_6,SIZE_6) ;
         strncpy(st_hpitch,buf+OFFSET_7,SIZE_7) ;
         strncpy(st_hroll,buf+OFFSET_8,SIZE_8) ;
         strncpy(st_hyaw,buf+OFFSET_9,SIZE_9) ;

/*  put in null terminators for sub-strings */

         st_julian_hour[SIZE_1] = '\0' ;
         st_julian_min[SIZE_2] = '\0' ;
         st_julian_sec[SIZE_3] = '\0' ;
         st_heading[SIZE_4] = '\0' ;
         st_vpitch[SIZE_5] = '\0' ;
```

```
        st_vroll[SIZE_6] = '\0' ;
        st_hpitch[SIZE_7] = '\0' ;
        st_hroll[SIZE_8] = '\0' ;
        st_hyaw[SIZE_9] = '\0' ;

/*  convert strings to floating point numbers */

        julian_hour = atof(st_julian_hour) ;
        julian_min = atof(st_julian_min) ;
        julian_sec = atof(st_julian_sec) ;
        time = julian_hour*3600. + julian_min*60. + julian_sec ;

        if (mgoodplat == 0)
          {
          time0 = time ;
          puplat->dtplat[mgoodplat] = 0. ;
          puplat->tzeroplat = time ;
          }
        else
          {
          puplat->dtplat[mgoodplat] = time-time0 ;
          }
        puplat->heading[mgoodplat] = atof(st_heading) ;
        puplat->vpitch[mgoodplat] = atof(st_vpitch) ;
        puplat->vroll[mgoodplat] = atof(st_vroll) ;
        puplat->hpitch[mgoodplat] = atof(st_hpitch) ;
        puplat->hroll[mgoodplat] = atof(st_hroll) ;
        puplat->hyaw[mgoodplat] = atof(st_hyaw) ;
        mgoodplat += 1 ;

        }
      }
    }

    *ngoodplat = mgoodplat ;
    *nbadplat = mbadplat ;

    free(st_julian_hour) ;
    free(st_julian_min) ;
    free(st_julian_sec) ;
    free(st_heading) ;
    free(st_vpitch) ;
    free(st_vroll) ;
    free(st_hpitch) ;
    free(st_hroll) ;
    free(st_hyaw) ;

    fclose(fpreadplat) ;
    return ;

}
```

```
/*****************************************************************************
*
*     Routine: trim.c
*
*     This module takes the x,y,amp vectors of default sizes
*     and trims them to their actual size
*
*     Doug DeProspo
*     08Feb95
*          Arete Engineering Technologies Corporation
*****************************************************************************
*/
#include "dipole.h"


void trim(double *xin,double *yin,double *ampin,int nout,double *xout,
          double *yout,double *ampout)

{

     int i ;

     for (i=0; i < nout ;i++)
          {
       xout[i] = xin[i] ;
       yout[i] = yin[i] ;
       ampout[i] = ampin[i] ;
       }

     return ;
}
```

This page is intentionally left blank

APPENDIX C

PRINTOUT OF A SAMPLE STD FILE

```
target\\
stdRec\
targetLocation 2958.16688: 8528.56590\
targetDepth -2.70\
*\
*\
targetSize L\
*\
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.17029: 8528.56987\
targetDepth -3.73\
*\
*\
targetSize L\
*\
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.17637: 8528.58217\
targetDepth -2.25\
*\
*\
targetSize L\
*\
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.18316: 8528.55934\
targetDepth -1.52\
*\
*\
targetSize L\
*\
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.18366: 8528.58818\
targetDepth -0.17\
*\
*\
targetSize L\
*\
```

```
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.17920: 8528.58620\
targetDepth -2.34\
*\
*\
targetSize L\
*\
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.19003: 8528.58673\
targetDepth -0.39\
*\
*\
targetSize M\
*\
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.19750: 8528.58453\
targetDepth -2.44\
*\
*\
targetSize L\
*\
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.17777: 8528.57492\
targetDepth -1.88\
*\
*\
targetSize L\
*\
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.19182: 8528.56666\
targetDepth -1.15\
*\
*\
```

```
targetSize L\
*\
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.17731: 8528.56076\
targetDepth -2.21\
*\
*\
targetSize L\
*\
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.19169: 8528.58518\
targetDepth -0.75\
*\
*\
targetSize L\
*\
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.19739: 8528.58297\
targetDepth -1.38\
*\
*\
targetSize L\
*\
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.18634: 8528.57384\
targetDepth -2.32\
*\
*\
targetSize L\
*\
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.17950: 8528.58503\
targetDepth -5.13\
```

```
*\
*\
targetSize L\
*\
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.18734: 8528.56385\
targetDepth -0.46\
*\
*\
targetSize L\
*\
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.19318: 8528.55927\
targetDepth -1.13\
*\
*\
targetSize L\
*\
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.17364: 8528.56666\
targetDepth -2.40\
*\
*\
targetSize L\
*\
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.19154: 8528.57526\
targetDepth -0.15\
*\
*\
targetSize M\
*\
*\
sizeConfidence  99\
*\
stdRec\
```

```
targetLocation 2958.18923: 8528.56706\
targetDepth -1.69\
*\
*\
targetSize L\
*\
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.19298: 8528.57590\
*\
*\
*\
*\
*\
*\
*\
*\
stdRec\
targetLocation 2958.19492: 8528.57007\
targetDepth -0.42\
*\
*\
targetSize M\
*\
*\
sizeConfidence  75\
*\
stdRec\
targetLocation 2958.18458: 8528.58093\
targetDepth -0.41\
*\
*\
targetSize S\
*\
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.17251: 8528.57256\
targetDepth  0.05\
*\
*\
targetSize S\
*\
*\
sizeConfidence  99\
```

```
*\
stdRec\
targetLocation 2958.17157: 8528.58263\
targetDepth -1.84\
*\
*\
targetSize L\
*\
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.17205: 8528.56061\
targetDepth -1.69\
*\
*\
targetSize L\
*\
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.17457: 8528.58641\
targetDepth -1.99\
*\
*\
targetSize L\
*\
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.18827: 8528.58547\
targetDepth -1.65\
*\
*\
targetSize L\
*\
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.17711: 8528.57716\
targetDepth -0.66\
*\
*\
targetSize L\
*\
```

```
*\
sizeConfidence  99\
*\
stdRec\
targetLocation 2958.19371: 8528.57564\
targetDepth -0.52\
*\
*\
targetSize S\
*\
*\
sizeConfidence  99\
*\
```

APPENDIX D

LOGIC DIAGRAMS FOR CLUSTER.C AND CURFIT.C

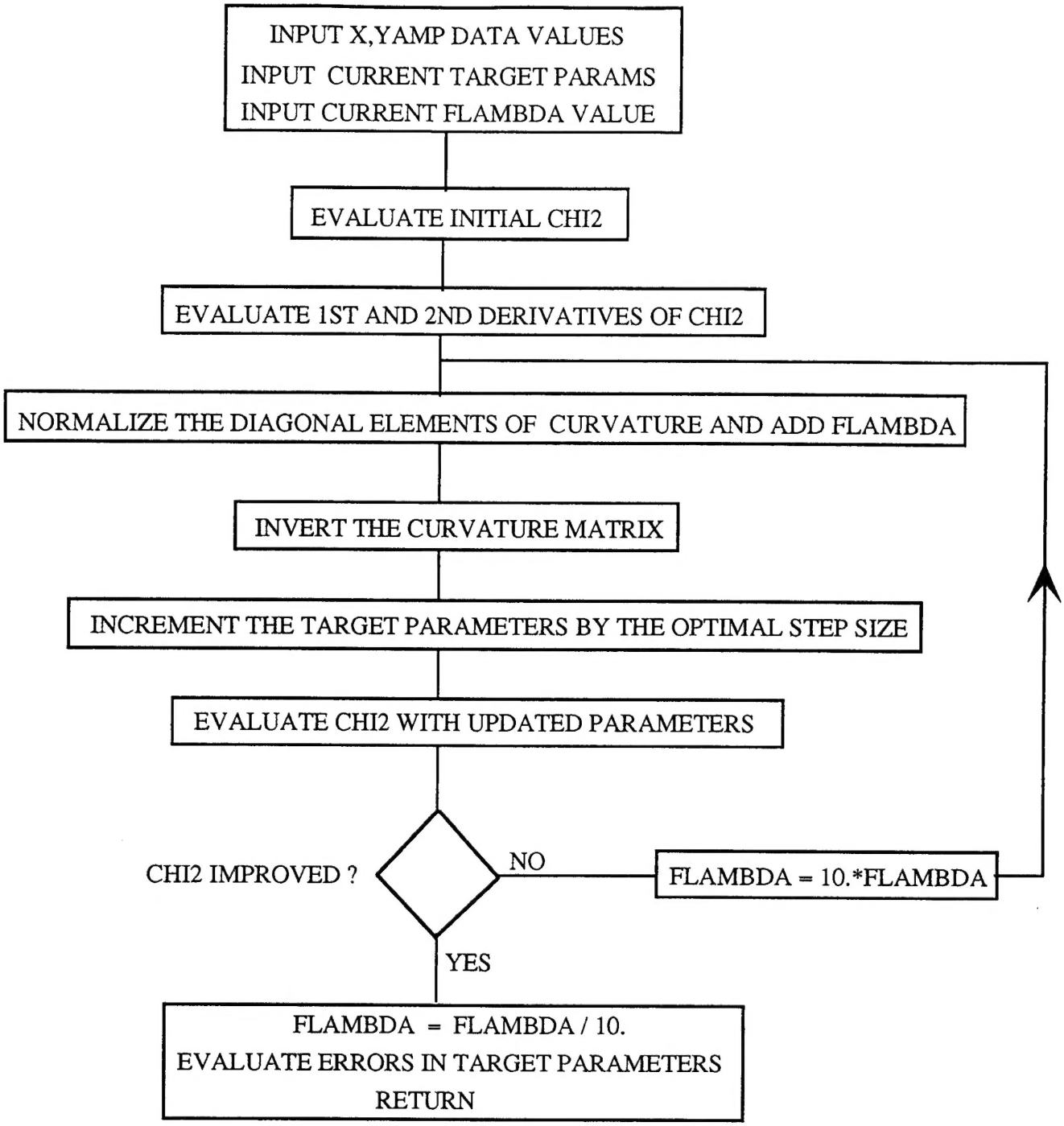


Figure D.1 Shown is a logic diagram illustrating the main processing elements in the maximum likelihood fitting routine *curfit.c.*

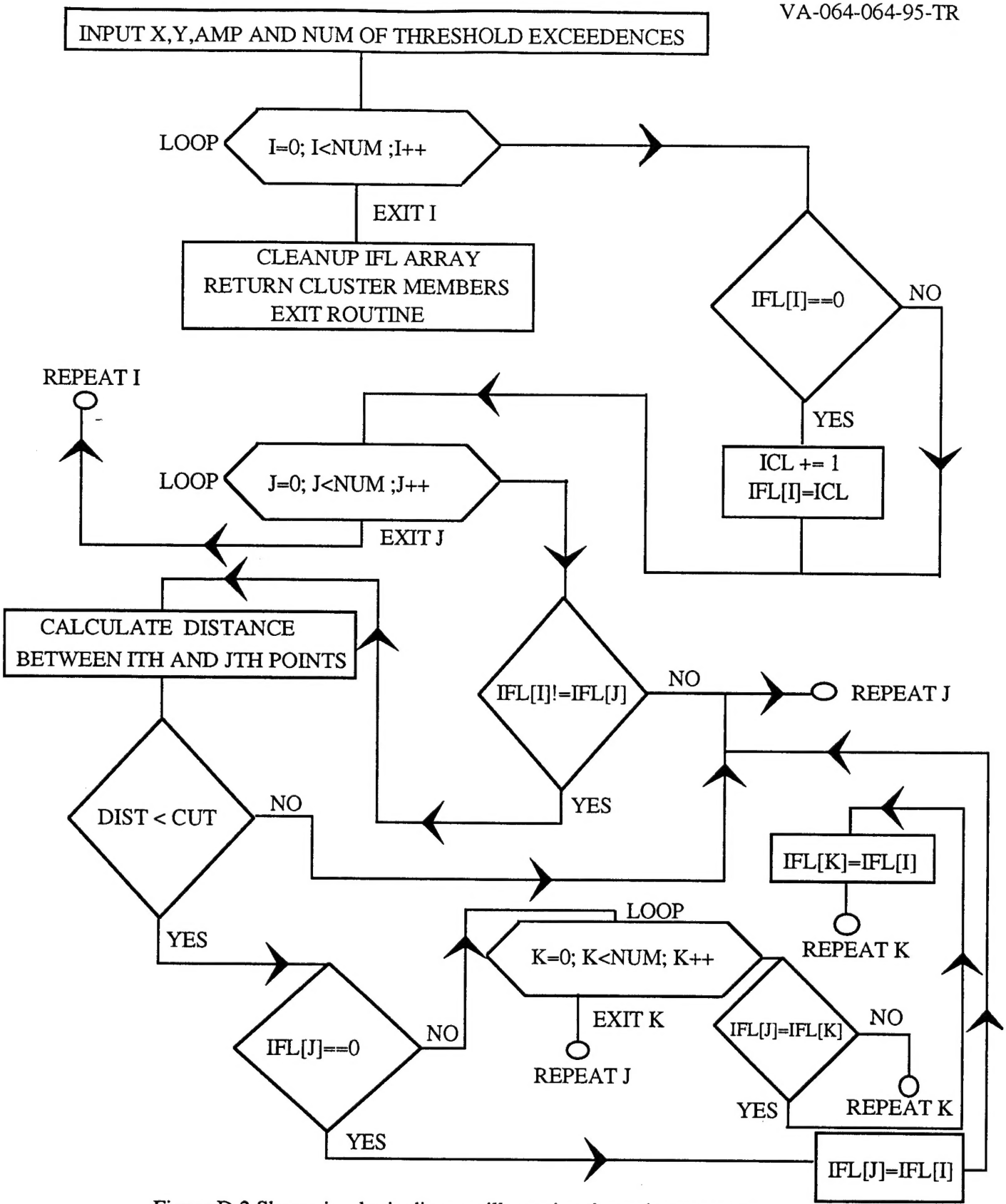


Figure D.2 Shown is a logic diagram illustrating the main processing elements in the detection algorithm *cluster.c.*